A1

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 7 : G06T 17/20 — A1

(11) International Publication Number: WO 00/19380

(43) International Publication Date: 6 April 2000 (06.04.00)

(21) International Application Number: PCT/US99/22194

(22) International Filing Date: 24 September 1999 (24.09.99)

(30) Priority Data:
09/163,075 29 September 1998 (29.09.98) US

(71) Applicant *(for GB only)*: SCHLUMBERGER LIMITED [NL/US]; 277 Park Avenue, New York, NY 10172 (US).

(71) Applicant *(for CA only)*: SCHLUMBERGER CANADA LIMITED [CA/CA]; 24th floor, Monenco Place, 801 6th Avenue, S.W., Calgary, Alberta T2P 3W2 (CA).

(71) Applicant *(for BF BJ CF CG CI CM GA GN GW ML MR NE SN TD TG ZA only)*: PRAD RESEARCH AND DEVELOPMENT N.V [NL/NL]; PO Box 812, De Ruyterkade 62, Willemstad, Curacao (AN).

(71) Applicant *(for BR only)*: SCHLUMBERGER SURENCO, S.A. [PA/VE]; Torre Humboldt, Piso 13, Avenida Rio Caura, Pargue, Humboldt, 1080 Caracas (VE).

(71) Applicant *(for AM AT AU AZ BY DE DK ES ID IE IT KG KR KZ MD MX NO NZ RU TJ TM only)*: SCHLUMBERGER TECHNOLOGY B.V. [NL/NL]; Parkstraat 83–89, NL–2514 JG The Hague (NL).

(71) Applicant *(for FR only)*: SERVICES PETROLIERS SCHLUMBERGER [FR/FR]; 42, rue Saint–Dominique, F–75007 Paris (FR).

(71) Applicant *(for all designated States except AM AT AU AZ BF BJ BR BY CA CF CG CI CM CN DE DK ES FR GA GB GN GW ID IE IT KG KR KZ MD ML MR MX NE NO NZ RU SN TD TG TJ TM ZA)*: SCHLUMBERGER HOLDINGS LIMITED [GB/GB]; P.O. Box 71, Craigmuir Chambers, Road Town, Tortola (VG).

(71) Applicant *(for CN only)*: SCHLUMBERGER OVERSEAS S.A. [PA/PA]; No. 8 Calle Aquilino de la Guardia, Panama City (PA).

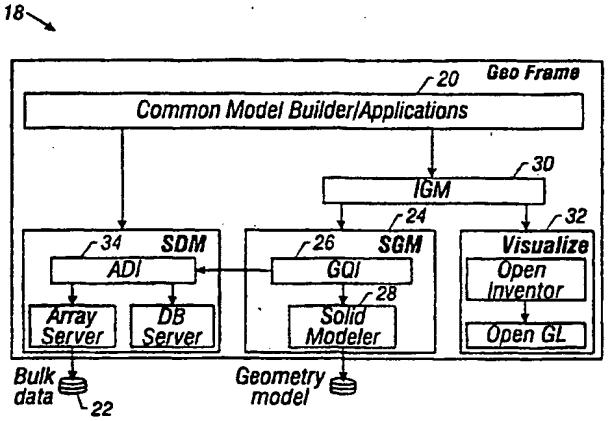
(72) Inventors: ASSA, Steven, B.; The Crescent 8, Storeys Way, Cambridge, Cambridgeshire CB3 0AZ (GB). HAMMERSLEY, Richard, P.; 3625 Duval Road, Number 1232, Austin, TX 78759 (US). LU, Hong–Qian; 6401 Deer Hollow Lane, Austin, TX 78750 (US).

(74) Agent: JANSSON, Pehr, B.; Schlumberger Austin Product Center, Intellectual Property Law Department, 8311 North FM 620 Road, P.O. Box 200015, Austin, TX 78720–0015 (US).

(81) Designated States: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, DE, DK, DM, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: MODELING AT MORE THAN ONE LEVEL OF RESOLUTION



(57) Abstract

A method, computer system and computer program are disclosed for representing a first surface at multiple levels of resolution. The first surface is partitioned into nodes with one or more boundaries, each level of resolution having a subset of the boundaries. A second surface may be classified against the first surface. Surfaces and the model may be decimated. Portions of the surfaces may be loaded from persistent memory on demand and removed when no longer required.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	**ES**	Spain	**LS**	Lesotho	**SI**	Slovenia
AM	Armenia	**FI**	Finland	**LT**	Lithuania	**SK**	Slovakia
AT	Austria	**FR**	France	**LU**	Luxembourg	**SN**	Senegal
AU	Australia	**GA**	Gabon	**LV**	Latvia	**SZ**	Swaziland
AZ	Azerbaijan	**GB**	United Kingdom	**MC**	Monaco	**TD**	Chad
BA	Bosnia and Herzegovina	**GE**	Georgia	**MD**	Republic of Moldova	**TG**	Togo
BB	Barbados	**GH**	Ghana	**MG**	Madagascar	**TJ**	Tajikistan
BE	Belgium	**GN**	Guinea	**MK**	The former Yugoslav Republic of Macedonia	**TM**	Turkmenistan
BF	Burkina Faso	**GR**	Greece			**TR**	Turkey
BG	Bulgaria	**HU**	Hungary	**ML**	Mali	**TT**	Trinidad and Tobago
BJ	Benin	**IE**	Ireland	**MN**	Mongolia	**UA**	Ukraine
BR	Brazil	**IL**	Israel	**MR**	Mauritania	**UG**	Uganda
BY	Belarus	**IS**	Iceland	**MW**	Malawi	**US**	United States of America
CA	Canada	**IT**	Italy	**MX**	Mexico	**UZ**	Uzbekistan
CF	Central African Republic	**JP**	Japan	**NE**	Niger	**VN**	Viet Nam
CG	Congo	**KE**	Kenya	**NL**	Netherlands	**YU**	Yugoslavia
CH	Switzerland	**KG**	Kyrgyzstan	**NO**	Norway	**ZW**	Zimbabwe
CI	Côte d'Ivoire	**KP**	Democratic People's Republic of Korea	**NZ**	New Zealand		
CM	Cameroon			**PL**	Poland		
CN	China	**KR**	Republic of Korea	**PT**	Portugal		
CU	Cuba	**KZ**	Kazakstan	**RO**	Romania		
CZ	Czech Republic	**LC**	Saint Lucia	**RU**	Russian Federation		
DE	Germany	**LI**	Liechtenstein	**SD**	Sudan		
DK	Denmark	**LK**	Sri Lanka	**SE**	Sweden		
EE	Estonia	**LR**	Liberia	**SG**	Singapore		

MODELING AT MORE THAN ONE LEVEL OF RESOLUTION

Background of the Invention

This application relates to representing computer models and more particularly to representing a geometry model at more than one level of resolution.

The accurate representation of subsurface topology can have a profound effect on the interpretation of a geoscience model. This is mainly due to the presence of material properties such as, for example, oil. For a more detailed introduction on the importance of topology, see United States Patent Application Serial No. 08/772,082, entitled MODELING GEOLOGICAL STRUCTURES AND PROPERTIES.

Imagine, for example, the situation of two compartments 10a, 10b separated by a sealed fault 12 (a sealed fault is an impermeable membrane that does not permit fluids to pass), as shown in Fig. 1. Due to the sealed fault there can be no flow of fluid from compartment 10a to compartment 10b. If both compartments were to contain oil it would be necessary to drill into both compartments to recover the oil.

Now suppose the sealed fault 12 is punctured, as shown in Fig. 2. There is now a free flow of fluid between compartment 10a and compartment 10b so it would be possible to drill just into one of the compartments to extract the oil.

Thus, having the correct topology in the geoscience model can have a profound effect on the finances of an oil-field development.

Another important concept in interpreting geoscience models presented graphically on a computer screen is the concept of multiresolution analysis, whereby an analyst can view an area of interest in the geoscience model at different resolution levels. There are many techniques that have been developed for multiresolution analysis of surfaces. The work falls into two principal categories. The first is to use wavelets. The second is to use edge contraction and edge flipping, which is sometimes called "topological editing".

Wavelets have found wide acceptance in image processing and recently have found application in surface representations. Typically, wavelets are used in a subdivision

fashion. A typical subdivision scheme uses quaternary subdivision. For example, as shown in Fig. 3, a triangle 14 may be subdivided into four triangles 16a-d. In the example shown in Fig. 3, the retessellation conforms to the subdivision scheme. That is, the retessellation of the triangle into four triangles conforms with the quaternary subdivision scheme. It can be imagined, however, that if a different local retessellation of the triangle is performed, it may not be clear how to rebuild the subdivision scheme, since the refinement may not conform to the subdivision scheme. For example, if the triangle 14 is retessellated as in Fig. 3b into triangles 17a and 17b, the retessellation does not conform to the quaternary subdivision scheme. There has been no work on the integration of wavelets and boundary representations.

Topological editing, or editing a mesh using the topological operations of vertex removal, edge contraction and edge flipping, can be used to build multiresolution surfaces. Some mesh building techniques build a history of topological operations which permits progressive and partial loading of the surface, but it is not clear how this history is modified if a triangle is refined. There has been no work on the integration of topological editing and boundary representations.

Summary of the Invention

In general, in one aspect, the invention features a method, computer system and computer program for representing a first surface at multiple levels of resolution. The first surface comprises zero or more zero-cells, zero or more one-cells and one or more two cells. The method is implemented in a programmed computer comprising a processor, a memory, a persistent storage system, at least one input device, and at least one output device. The method and a model are stored on a computer-readable media and the method represents the model on one of the output devices. The method comprises partitioning the first surface with one or more boundaries, each level of resolution having a subset of the boundaries.

Implementations of the invention may include one or more of the following. The first surface may be partitioned into n_i nodes at resolution level-i using the level-i subset of boundaries. Each level-i+1 node may be associated with a unique level-i node. Each level-i node may be associated with the level-i+1 nodes associated to the node. Each level-i node may be associated with a subset of vertices that are critical at resolution level i. Assuming level d is the deepest level of resolution and the first surface is divided into simplices, each node at resolution level d may be designated a leaf node, each simplex may be associated to a unique leaf node, and each leaf node may have associated with it the simplices associated to that leaf node.

A level-i node may have associated with it the list of simplices which is the union of all simplices associated with the level-i+1 nodes associated to the level-i node. The subset of boundaries for each node may be assigned to be the boundary of the union of the simplices associated with that node. The nodes may form an original tree and each node may be assigned a unique key. Each vertex in a leaf node may be assigned the key corresponding to that leaf node.

The representation of a second surface may be stored in the computer-readable media, the second surface having nodes, leaf nodes, vertices, critical vertices and simplices. It may be determined which leaf nodes of the first surface intersect the leaf nodes of the second surface and the intersecting simplices from the first and second surfaces from the simplices associated to the intersecting leaf nodes.

Each node except the leaf nodes may have a subtree, and the original tree may be split into new trees and each new tree may be associated with a new cell. The subtrees of the original tree which have no intersecting leaf nodes may be identified with one of the new cells. The simplices of the first surface may be split along the intersection curve. New simplices may be formed by tessellating the split simplices to respect the macro-topology of one-cells and zero-cells passing through the original simplices. A new tree may be built for each new cell and each new simplex may be assigned to the leaf node of the tree created for the new cell to which the new simplex belongs. For each leaf node of each new tree, each simplex in the original tree which is connected to a new simplex in the new tree leaf node and which lies in the same tree leaf node as the new simplex may be migrated. The neighbors of a tree node may be determined by finding all the keys of all the critical vertices in the node. The coarsest level node which is an ancestor of a key from the critical vertices in the migrated tree nodes and has not been split or migrated may be determined and that node may be migrated to the new tree.

A complete node front of the tree of the first surface and a collection of vertices on a boundary of the first surface may be defined. A list of critical vertices from the tree nodes of the complete node front may be built. Those vertices identified to one- or zero-cell vertices may be removed from the list and all zero-cell vertices from the model which lie in the first surface and the defined collection of one-cell vertices may be added to the list. The collection of one-cell edges may be recorded. The surface may be tessellated to respect the list of vertices and the recorded one-cell edges.

The subset of vertices on the boundary of the first surface which are also on the boundary of the second surface may be required to be the same as the subset of vertices on the boundary of the second surface which are also on the boundary of the first surface.

A geometrical representation of the first surface may be maintained in a persistent storage.

A bounding box for each node may be stored on a persistent storage device, and for each node a list of critical vertices associated with that node may be stored on the persistent storage device. Storing the list of critical vertices may comprise storing a vertex descriptor for each vertex, storing a parameter value for each vertex, and storing an image value for each vertex. That portion of the first surface required may be loaded on demand from persistent storage and that portion of the first surface not required may be removed from memory. A tree node may be loaded from persistent storage on demand and removed from memory when it is no longer required. Simplices associated with a tree leaf node may be loaded from persistent storage on demand and removed from memory when no longer required.

In general, in another aspect, the invention features a method, computer system and computer program for representing a first surface at multiple levels of resolution. The method is implemented in a programmed computer comprising a processor, a data storage system, at least one input device, and at least one output device. The method and a model are stored on a computer-readable media and the method represents the model on one of the output devices. The method comprises storing a grid representation of the first surface, the grid representation being made up of grid cells, forming a mesh representation of a portion of the first surface by triangulating a subset of the grid cells, and inserting the first surface into the model.

Brief Description of the Drawings

Figs. 1, 2, 5a, 5b, 6a, and 35-41 are representations of models.

Figs. 3a and 3b are representations of prior art multiresolution analysis schemes.

Fig. 4 is a block diagram.

Fig. 6b is a representation of the geometry of a model.

Fig. 6c is a topology graph.

Figs. 7a-7d illustrate simplices.

Figs. 8a, 8b and 8c illustrate a simplicial complex, a simplicial complex with a

hole and an illegal simplicial complex, respectively.

Fig. 9 illustrates a non-manifold triangulated surface.

Figs. 10a-10c illustrate classification of one surface into another.

Figs. 11a-11d illustrate the need for coherent models to perform editing.

Figs. 12a-12c illustrate making a model coherent.

Figs. 13a-13c illustrate grid representations.

Fig. 14 illustrates a mesh representation of a surface.

Fig. 15 is a table of rendering times and rates.

Figs. 16a-c illustrate cracking caused by approximation.

Figs. 17a-b illustrate cracking caused by approximation.

Figs. 18a-b illustrate bubbling caused by approximation.

Figs. 19a-b illustrate bubbling caused by approximation.

Figs. 20a-b illustrate the two possible triangulations of a grid cell.

Fig. 21 is a graphical representation of a quadtree.

Fig. 22 is a geometrical representation of a quadtree.

Fig. 23 is a table showing the size of the different components of a quadtree.

Figs. 24a-b illustrate the effect of removing a non-critical vertex through approximation.

Figs. 25a-b illustrate the effect of removing a critical vertex through approximation.

Fig. 26 illustrates internal critical vertices.

Fig. 27 illustrates external critical vertices.

Figs. 28a-b, 29a-b illustrate approximation of a surface while respecting critical vertices.

Figs. 30a-e and 33 are flow diagrams.

Fig. 31 represents a quadtree with leaf nodes at a fixed depth.

Fig. 32 illustrates parent critical, edge critical and sub-critical vertices.

Fig. 34 is a table listing common surfaces and their Euler Characteristics.

Fig. 35 illustrates embedded disc obstructions that may be found in an external boundary.

Fig. 36 illustrates a method for removing a particular type of obstruction.

Fig. 37 illustrates a method for determining the contour formed from the set of all vertices connected to a recently defined contour.

Fig. 38 illustrates projecting an upper contour of a truncated square-based pyramid onto the interior of a square defined by the lower contour of the pyramid.

Fig. 39 illustrates square based pyramids with contours meeting in a non-manifold manner.

Fig. 40 illustrates contours being the boundary of an embedded disk.

Fig. 41 illustrates pyramids formed with triangle caps and square bases.

Description of the Preferred Embodiments

A geoscience environment 18, called GeoFrame, comprises an application framework 20 which comprises a shareable database 22, a geometry modeler (SGM) 24, a geometry query interface (GQI) 26, a solid modeler 28, an interactive geometric modeling library (IGM) 30 and a renderer (Visualize) 32, as shown in Fig. 4 and explained in United States Patent Application Serial No. 08/772,082, entitled MODELING GEOLOGICAL STRUCTURES AND PROPERTIES, incorporated by reference.

An application accesses shareable database instance data using the Application Data Interface (ADI) 34. The ADI provides a programmatic interface to the shareable database 22. The geometry modeler (SGM) 24 describes the part of the shareable database 22 specific to 3D geometrical modeling. An application accesses SGM instance data using the GQI 26.

The GQI provides a procedural interface to access the SGM. The GQI also provides a stable interface to a low-level non-geologic solid modeler 28 and adds extra

organization to the low-level geometric representations to aid geological understanding.

The GQI is implemented on top of a low-level geometric solid modeler 28, such as SHAPES™ from XOX.

The Interactive Geometric Modeler (IGM) 30 provides visualization services on behalf of and a graphical interface to the GQI. An application normally receives human instruction through the IGM.

The IGM is implemented on top of a low-level rendering engine 28, such as a combination of OPENINVENTOR and OPENGL.

The Common Model Builder 20 is a toolkit that brings together the 3D geometric modeling components. It integrates data management services, geometric modeling and graphical interfaces for human input. The Common Model Builder provides an abstract framework upon which a 3D geoscience application can be built.

The GQI clearly distinguishes between the notions of topology and geometry. Broadly, topology refers to the connectivity between components in the model and generally refers to "macro-topology", described below. Geometry refers to the actual point-set representation of a particular component, for example.

Macro-topology carries the relationships between the major topological components of a model. The major topological components refer to points, edges (curves), faces (surfaces), and subvolumes. For example, macro-topology would answer a question such as "which surfaces bound a particular volume?" or "which surfaces lie within a particular volume?". Macro-topology does not need to consider the underlying geometry of surfaces and can work with many different types of geometrical representations, for example NURBs and meshes. Macro-topology is implemented in the SHAPES solid modeler 28.

In the terminology of the GQI, macro-topology is represented by cells. A cell is a path-connected subset of Euclidean space of a fixed dimension. Path-connected means any two points in the cell can be connected by a path in the cell. The dimension refers to

the dimension of the geometry of the cell, a 0-cell is a point, a 1-cell is an edge (curve), a 2-cell is a face (surface) and a 3-cell is a volume. For example, areas 36 and 38 are distinct cells separated by fault 40, as shown in Fig. 5a. Curve 42 illustrates the path-connected character of cell 38. Similarly, as shown in Fig. 5b, the line segments 44 between intersection points and the intersection points 46 themselves are cells.

The topology of a geometric model is the set of all cell-cell connectivity relationships. The cell-cell connectivity relationships can be represented in a graph in which the arcs of the graphs represent connectivity and the nodes represent cells, as shown in Figs. 6a-c. In a cellular model, cells of dimension n are connected to boundary cells of dimension n-1 and vice versa. For example, in the topology of a box 48, shown in Fig. 6a, the area cell 50 is connected to its four bounding edge cells 52a-d. A single cell can act both as a boundary and as a region. For example, a surface can bound a subvolume, but can itself be bounded by a set of curves. The 1-cells 52a-d, in Fig. 6b, are both subregions (bounded by zero-cells 54a-d) and boundaries (of area cell 50). These relationships can be represented graphically, as shown in Fig. 6c. Node 56, corresponding to area cell 56, is connected to nodes 58a-d, representing 1-cells 52a-d, respectively. The connection between the nodes is represented by the arcs between them. 1-cells 58a-d are connected to zero-cells 60a and 60d, 60a and 60b, 60b and 60c, and 60c and 60d, respectively. Cells which are contained in higher-dimensional cells but do not split them (such as fault 40 in Fig. 5a) are said to be "embedded".

Geometry is the point-set description of a topological object. For example, a geometrical query would be "which is the closest point on the surface to a given point?".

All geometry to be considered here is piecewise linear. For convenience, a brief review of the definition and topological properties of a simplex, simplicial complex, and triangulated surface or mesh is included below. In short, the piecewise linear geometry of an 1-cell or edge is a polyline, for a 2-cell or face it is a triangular mesh and for a 3-cell it is a tetrahedral mesh.

Let $a_0,...,a_r$ be a collection of r+1 linearly independent points in R^n with $r \leq n$. The r-dimensional simplex $\sigma = \langle a_0,...,a_r \rangle$ is the set of all points in R^n such that

$$\sigma = \left\{ x \in R^n \mid x = \sum_{i=0}^{r} t_i \bullet a_i, t_i \geq 0, \sum_{i=0}^{r} t_i = 1 \right\}$$

The r-tuple $(t_0,...,t_r)$ is the barycentric coordinate of x. The $\{a_i\}$ are the simplex's corner points.

A 0-simplex is a point 62, illustrated in Fig. 7a, and a 1-simplex is the line segment 64 joining, for example, a_0 to a_1, as shown in Fig. 7b. A 2-simplex is a filled triangle in 3-space 66, the triangle being defined by the three corner points, as shown in Fig. 7c. Similarly, a 3-simplex is a filled tetrahedron 68 defined by the four corner points, as shown in Fig. 7d.

Let $\sigma = \langle a_0,...,a_r \rangle$ be an r-simplex and let q be an integer with $0 \leq q \leq r$. Now choose q+1 distinct points from $a_0,...,a_r$, say $a_{i0},...,a_{iq}$, these q points define a q-simplex which is called a q-face of σ.

An n-simplex has n+1 n-1-faces. These faces are exactly the faces expected, that is the 0-faces of a 1-simplex are the end points, the 1-faces of a 2-simplex are the bounding lines, the 2-faces of a 3-simplex are the triangular faces of the tetrahedron.

A simplicial complex, K, is a finite collection of simplices in some R^n satisfying

1. If $\sigma \in K$, then all the faces of σ belong to K
2. If $\sigma, \tau \in K$ then either $\sigma \cap \tau$ = null or $\sigma \cap \tau$ is a common face of σ and τ.

Examples of legal simplicial complexes include a simplicial complex with vertices, e.g. 70, and simplices, e.g. 72, as shown in Fig. 8a, and a simplicial complex with a hole 74, as shown in Fig. 8b. Overlapping simplices, e.g. 76, create an illegal simplicial complex, as shown in Fig. 8c

The "dimension" of a simplicial complex is –1 if K = null, and the maximum dimension of the simplices of K otherwise.

Given a collection of simplices C from the simplicial complex K, the collection C defines another simplicial complex by considering all the faces of C:

$$SimpComp(C) = \{\sigma \in K | \sigma \text{ is a face of a simplex in } C\}$$

Let K be a simplicial complex and define:

$$\Delta_s^r(K) = \left\{ \tau \middle| \begin{array}{l} \tau \text{ is an } s-\text{simplex} \\ \tau \text{ is the face of exactly } r \text{ distinct } s+1 \text{ simplices} \\ \tau \text{ is not the face of any } s+2 \text{ simplices} \end{array} \right\}$$

and

$$\delta_s^r(K) = SimpComp(\Delta_s^r(K))$$

A pseudomanifold, M, is a simplicial complex such that

1. M is homogeneously n-dimensional. That is, every simplex of M is a face of a n-simplex of M.
2. Every (n-1)-simplex of M is a face of at most two n-simplices.
3. If σ and σ' are two distinct n-simplices of M, then there exists a sequence $\sigma_1,\ldots,\sigma_k$ of n-simplices in M, such that $\sigma_1 = \sigma$, $\sigma_k = \sigma'$ and σ_i meets σ_{i+1} in a (n-1)-face for $1 \le i \le k-1$.

A triangulated surface or mesh is a two-dimensional pseudo-manifold. The presence of a singular vertex 78, as shown in Fig. 9, means that the triangulated surface 80 is not a manifold, but it is a pseudo-manifold.

For an n-homogenous simplicial complex K, i.e. every simplex is a face of some n-simplex, the following can be seen to hold:

$$\delta_0^0(K) = \delta_1^0(K) = \ldots = \delta_{n-1}^0(K) = null$$

The boundary of a simplicial complex K of dimension n, is given by the collection of simplices:

$$\delta(K) = \delta_0^1(K) \cup \delta_1^1(K) \cup \ldots \cup \delta_{n-1}^1(K)$$

For an homogenous simplicial complex of dimension n, the boundary reduces to:

$$\delta(K) = \delta_{n-1}^1(K)$$

The boundary of an homogenous simplicial complex K of dimension n is itself an homogenous simplicial complex of dimension n-1. However, the boundary of a pseudo-manifold is not necessarily a pseudo-manifold.

Denote by Simps(v) the set of 2-simplices which connect to the vertex v in a triangle mesh.

Denote by Verts(S) the set of vertices forming the corners of the 2-simplex S.

Any surface representation must be able to pass geometry to the underlying geometry engine. The SHAPES geometry engine uses micro-topology for this purpose.

Micro-topology is an example of a mesh representation. The mesh representation is an integral part of the surface representation. It is sufficient for the mesh representation to support basic navigation queries, such as, "which simplices use a vertex?" and "which vertices are used by a simplex?". Using these two queries it is possible to navigate from a simplex to a vertex and then from this vertex back to another simplex. In this way it is possible to determine connected components in a model. Micro-topology supports these types of queries.

An additional requirement, which is imposed by the SHAPES geometry engine, is that all intersection curves must be supported by micro-topology.

For the purposes of the surface representation it is necessary to know which vertices lie on the boundary of a surface. This information is maintained by the SHAPES geometry engine and permits efficient queries identifying whether a vertex is identified to a 1-cell vertex or a 0-cell vertex.

An additional requirement imposed by the SHAPES geometry engine is that all vertices which are identified with a 1-cell or 0-cell vertex and all boundary simplices are retained in core memory at all times.

One of the algorithms provided by the GQI is classification. Given two cells, A and B, classification subdivides the respective point sets into an inside part, an outside part, and a part on the boundary of the other. In set theoretic terms, the union of A and B breaks into three disjoint components, the part of A which is not in B (A\B), the part of B which is not in A (B\A) and the intersection of A and B:

$$A \cup B = A\backslash B \cup B\backslash A \cup (A \cap B)$$

$$A\backslash B \cap B\backslash A = \text{null set}$$

$$A\backslash B \cap (A \cap B) = \text{null set}$$

$$B\backslash A \cap (A \cap B) = \text{null set}$$

When performing the classification, the GQI identifies the connected components or cells in the model, connects these cells along their shared boundaries, throws away the original cell definitions and builds new cell definitions for the connected components. The outcome is an Irregular Space Partition (ISP).

For example, as shown in Figure 10a, if an earth model 82 initially comprising a volume 84 and two horizons 86 and 88 has inserted into it a fault 90, shown in Fig. 10b, the classification proceeds as follows.

Compute the intersections of shapes and generate cells representing the intersection geometries. In Figure 10c, the intersection geometry is represented by the heavy dot intersection points 92.

Split cells are sub-divided by lower-dimensional cells. In Fig. 10c the horizons 86 and 88 intersect fault 90. The fault 90 ceases to exist and is replaced by cells 94, 96 and 98. Similarly, the combination of horizons 86 and 88 and cell 96 splits area 84 in two. Thus, area 84 ceases to exist and is replaced by areas 100 and 102.

Classification is a macro-topological operation which is implemented using micro-topological functionality. All surfaces that provide micro-topology must be able to support the micro-topological interface required by the macro-topological interface to perform classification.

"Coherency" occurs if the geometry of the cells agrees in all dimensions. The need for coherency is illustrated, for example, in Fig. 11a-d, where a volume of interest 104 is classified with a saltdome 106 and classified with a horizon 108a-c. That part of the horizon 108b which is interior to the saltdome and geologically would not be present is then removed.

To remove the interior surface 108b requires breaking up the horizon into two components, an exterior part 108a, 108c and an interior part 108b. To do this requires two distinct geometrical parts, the interior part and the exterior part. Once the geometry has been identified it is then a matter of removing the geometry and recomputing the macro-topology. This operation can be performed if the model is coherent.

An ISP is coherent if the geometry of the cells agrees at all dimensions (i.e., if it is a homogenous simplicial complex, as defined above). Effectively, this means the 0-cell 0-simplices and 1-cell 1-simplices are faces of the 2-cell 2-simplices.

Returning to Figs. 11a-d, to enable the removal of the interior surface 108b requires a geometrical representation of the surfaces which is coherent. In Figs. 12a-c, a cross-sectional view of an example of classification followed by coherency is given. A cell 110 consists of two simplices 112, 114, as shown in Fig. 12a. Cells 112 and 114 are coherent because their geometry agrees at all dimensions. Volume 110 is coherent. Cell 110 is classified against surface 116 which results in two new cells, cell a = {118, 122} and cell b = {120, 124}. Cells a and b are not coherent because their boundaries are not faces of simplices.

To correct this, simplices 118, 120, 122, and 124 are split to respect the boundaries of cells a and b, into simplices 126a-d, 128a-c, 130a-c and 132a-d, respectively.

This makes cells a and d coherent.

There are two stages to making a model coherent. The first requires a re-tessellation of the surface in the neighborhood of the intersection curves in order to ensure the model is coherent. The second is to perform a migration, which involves collecting together the simplices into connected components or cells and identifying each connected component with an appropriate cell in the ISP.

The retessellation can introduce tolerancing issues, because in performing the retessellation it is possible that degenerate triangles are produced. For example, a degenerate triangle may have a poor aspect ratio, or perhaps a small area. It is the responsibility of the underlying geometry engine to handle the degenerate triangles, which is typically done by merging degenerate triangles into neighboring non-degenerate triangles.

In the GQI, it is necessary for the surface representation to be "parameterized". This means each surface has an "image space" and a "domain space". The image space has to be the same for all surfaces and is typically three-dimensional Euclidean space. The domain space for a surface is typically a rectangle in the plane and as classification proceeds the domain is restricted to subsets of the rectangle.

There are two discrete surface representations which are of interest, grid and mesh. In any surface representation there are two important aspects, the topology and the geometry. Topology is used in its generic sense and refers to how the surface is connected, for example, which triangles connect to which triangles. The geometry specifies the actual position of the surface. The primary distinction between grid and mesh is grid represents topology implicitly, mesh represents topology explicitly.

There are several types of grid. They share a common feature which is that topology is represented implicitly. They differ in how the geometry is represented. The topology is represented implicitly as two integer extents giving the number of grid cells in each direction.

The most compact form of grid is a "regular grid", as illustrated in Fig. 13a. For a regular grid it is only necessary to store an origin, step values for the grid points, the number of grid points and the height values for each grid point. A regular grid has a number of drawbacks. In particular, all of the grid cells are of fixed size and only height fields can be represented.

A slightly more general form of grid is a "rectilinear grid". For a rectilinear grid the grid cell sizes can vary along each axis, as illustrated in Fig. 13b. As with regular grids, however, only height fields can be represented.

The most general type of grid is a "structured grid", as illustrated in Fig. 13c. As with regular and rectilinear grids, the topology is represented implicitly by two integer extents giving the number of grid cells in each direction. The geometry is represented explicitly by maintaining a three dimensional point coordinate for each grid point. A structured grid differs from regular and rectilinear grids because it is possible to represent multi-valued height fields. It is also more adaptive than regular and rectilinear grids allowing the grid cell size to vary across the whole grid.

Structured grids have the following characteristics: (1) they are compact; (2) they can represent multi-valued height fields; (3) topology is represented implicitly in that grid cell neighbors are given by increments and decrements of an indexing function; (4) the grid index is a natural parameterization; and (5) they are difficult to edit.

The major drawback of all grid representations is the inability to topologically edit the grid. For example, it is easy to move a vertex in a structured grid by replacing the coordinates. It is difficult to insert a new vertex into the grid, which would require regenerating the grid indices for the surface.

The greatest advantage of a mesh representation is the ability to represent irregular geometries. A mesh 142 with an irregular hole 144 can be represented very simply, as shown in Fig. 14. Where a surface is rapidly changing the mesh can be very fine, and in large flat regions the mesh can be very coarse. It is also very easy to edit a mesh. For

example, if a new vertex needs to be inserted, it is possible to retriangulate the surface in the neighborhood of the vertex.

Mesh has the following characteristics: (1) triangles are of variable size; (2) they can represent multi-valued height fields; (3) topology is represented explicitly; (4) locally editable, for example refinement and coarsening; (5) not necessarily parameterized.

Meshes can represent more general surfaces than grids. Compared to a grid, however, a mesh incurs a memory and performance cost because the topology has to be represented explicitly. However, because a mesh is irregular it can easily model multi-valued surfaces and surfaces which are rapidly changing in one area and flat in another.

When creating any surface representation, the sampling size can have a profound effect on memory usage. For example, if the sampling along each axis of a grid is doubled, the memory usage will be quadrupled. This suggests the need to selectively load portions of the model in core memory.

Furthermore, it is easy to build a geometric model which can reside in core memory, but which overwhelms even the most powerful graphics hardware. This suggests the need to be able to decimate or subsample the surfaces in an efficient manner for rendering.

Consider the following scenario. A geologist wants to create and edit on a workstation a 3D geometric model made up of meshes containing on the order of 250,000 simplices per surface and to have on the order of 50 surfaces. Fig. 15 records the rendering time in seconds on an SGI INDIGO^2 EXTREME with a GU1-Extreme graphics card and 128 Mbytes core memory for a set of N x N rectangular grids for a range of N. Each N x N grid contains $2(N-1)^2$ simplices. A grid's nodes are evenly spaced in two dimensions, e.g. x and y axes, but not in the third dimension, e.g. z. An optimization tool by Open Inventor (ivfix) was used to optimize the output.

In this example, the rendering rate varies with grid size, but is roughly 200,000 simplices per second. For good interactive response, rendering times on the order of 0.1

second are required, because a geologist-user wants to be able to travel interactively around the model. In the example of an SGI INDIGO^2 EXTREME, this speed is achieved for a 160 x 160 grid. In general, this figure will be highly dependent on the particular graphics hardware installed on a particular machine.

Increasing the power of the graphics hardware can significantly improve the rendering performance, but even this is of limited utility. Imagine a window on a computer of 500x500 pixels and hardware that is capable of rendering 50,000 triangles in 0.1 seconds. The total number of pixels is 250,000 that gives an average area of 5 pixels for each triangle. If the power of the rendering hardware is increased five-fold, a triangle will have an average area of 1 pixel. Beyond this point, increasing the power of the graphics hardware, and hence permitting more triangles to be rendered, will not significantly improve the quality of the screen image.

Real world surfaces are typically non-uniform. In one region a surface may be varying rapidly, in another it may be almost flat. This suggests the need for allowing the user to selectively choose different levels of detail in different regions of the model.

A geologist uses a 3D geometrical representation much like a microscope, in the sense that she wants to zoom in and zoom out in relatively small regions in a controlled manner. This usage is similar to that supported by terrain visualization implementations.

For visualization it would beneficial to draw parts of the model in the far field of view in a coarse representation, the parts of the model in the near field of view in detail.

After classification and coherency procedures have been performed, a valid ISP has been built. Typically, the model is too detailed to be viewed interactively on hardware that exists at present so the amount of detail from the model is reduced or "decimated". To avoid unnecessary artifacts in the model it is important that the decimated model respect the topology of the underlying ISP, as discussed above. This means all features of the boundary representation must be represented. The two effects that must be avoided are cracking and bubbling as discussed below.

There are two basic changes that can happen in the topology of a model. These two changes can happen in two places in the topology, the first is within a surface itself, the second in the boundary representation.

Cracking is often seen as a hole where there was no hole before, as shown in Figs. 16a-c. Two approximations are shown of a half-cylinder 146, shown in Fig. 16a. The first approximation 148, illustrated in Fig. 16b, exhibits a hole 150 in the surface, while the second approximation 152, shown in Fig. 16c, does not.

Cracking can also occur in the boundary representation of the ISP, as shown in Figs. 17a-b. An ISP 154, shown in Fig. 17a, has an intersection curve 156 splitting a surface into two cells 158 and 160. When the ISP is decimated as shown in Fig. 17b (with cell 158 being approximated, but cell 160 not being approximated), the intersection curve on cell 158 has been approximated by a straight line 162, the intersection curve on cell 160 has not been approximated. Due to the different approximations, the boundaries of cells 158 and 160 are no longer coincident and a hole 164 appears.

Bubbling is where intersections occur where there were none before decimation. As with cracking this can occur within the surface and within the boundary representation. Bubbling within a surface can occur where a non-self-intersecting surface becomes self intersecting, as shown in Fig. 18a-b. In Fig. 18a, a cross-sectional view is given of a surface 166. Vertex 168 is dropped which causes the surface to become self-intersecting at point 170, as shown in Fig. 18b, introducing a bubble 172.

Bubbling can occur in a similar fashion to the cracking that occurred in Figs. 17a-b, as shown in Figs. 19a-b. Fig. 19a shows a cross-section of an ISP 174 comprising of two surfaces 176 and 178. In a decimated version of the ISP 180, shown in Fig. 19b, surface 176 is decimated by dropping vertex 182, while surface 178 remains unchanged. This introduces new intersections 184 and 186 between surfaces 176 and 178 that did not intersect in the original ISP 174, producing bubble 188.

A new hierarchical surface representation is able to represent a surface that is

initially imported as a structured grid and is then edited by the use of classification and coherency. The representation supports the micro-topological interface required by the SHAPES geometry engine for a surface to be used in classification. The surface also supports adaptive decimation algorithms that prevent cracking and bubbling. The architecture is a hybrid-grid mesh. This means wherever possible a grid is used, but, when necessary, mesh regions overlaying the grid are used. This has many advantages, for example, being able to tune algorithms to make use of a grid representation when the surface is a grid. But, when greater flexibility is required, the grid representation can be exchanged for a mesh representation.

Typically, surfaces are imported as grids. However, the geometry engine can only perform geometric calculations with mesh. The hybrid grid-mesh representation neatly solves this problem, because areas of the grid can be dynamically converted to mesh thereby supporting general topological and geometrical editing. But, the efficient grid representation can be used where such general editing is not required. Effectively, the underlying geometry engine is fooled into believing the surface is a mesh. Furthermore, since it is possible to identify which areas are grid and which are mesh, algorithms can be optimized to make use of the grid structure whenever possible. An example of tuning when the surface is a grid is encoding the particular triangulation of the grid. A grid cell has two possible triangulations 190 and 192, as shown in Figs. 20a-b.

The triangulation of the grid can be stored in a bit vector, each bit representing the chosen triangulation of a particular grid cell. In this way it is not necessary to maintain the triangulation of the grid explicitly as simplices.

In general, the grid can be thought of as being the background and the mesh the foreground. Since the surface maintains its grid representation in the background it is possible to discard the mesh foreground at any time and recover the grid background. At any time a portion of the grid can be turned to mesh by reading the triangulation bit-vector and dynamically building simplices.

The hybrid architecture maintains the flexibility to provide the irregular refinement of a grid. This is important, for irregular refinement is required for efficient classification and coherency algorithms.

For the purposes of the implementation, a quadtree has been used to provide a multiresolution hierarchy. A quadtree was chosen because of its geometrical relationship to sub-sampling in grids.

A quadtree is a tree with nodes, each of which has four children, except for the leaf nodes, which have no children. Any quadtree can be drawn graphically, as shown in Fig. 21, or geometrically, as shown in Fig. 22. Every node of the tree has a unique depth and can be assigned a unique key.

The key of a quadtree node is chosen to provide the following functionality.

- A compact and efficient way to dereference quadtree nodes.
- A linear ordering for the quadtree nodes.
- An efficient way to compute the depth of the key.
- An efficient way to compute whether a key is an ancestor of another key.
- An efficient way to compute whether a key is a descendant of another key.
- A way to compute the ancestor keys of a key.
- A way to compute the descendant keys of a key.

Efficient in this context means a small number (typically <5) of bitwise Booleans together with bit shifts and arithmetic operations. The implementation of the quadtree key can be done by using pairs of bits to hierarchically identify the child at each level and representing the depth as an integer. Let ceil(x) be the smallest integer greater than or equal to x. Using the implementation of the quadtree key described above, the number of bits required to represent a tree of depth d is given by $2d+\text{ceil}(\log_2(d+1))$, where the factor 2d is the mantissa (2 bits are required to identify each child at each depth), and the factor $\text{ceil}(\log_2(d+1))$ is the number of bits required to represent the integer value d. Thus a 32 bit

key can represent a tree of depth 14, and as shall be seen this is more than adequate for current needs.

The mantissa is implemented using bit interleaving.

In Fig. 23 the size of the different components of a quadtree are given for different depths of the quadtree. It can be seen a tree of depth 10 can support a grid of size 1000 by 1000, furthermore, the key is comfortably maintained in 32 bits.

The following definitions will be used:

- The number of elements (cardinality) in a collection C will be denoted by Card(C).
- The key of a quadtree node N will be denoted by Key(N).
- The quadtree node of a key K will be denoted by Node(K).
- The depth of the key K will be denoted by Depth(K). Without loss of generality, the root key has depth 0.
- The ancestor key at depth i of the key K will be denoted by $Ancestor_i(K)$. The function is defined for $i \leq Depth(K)$ with $Ancestor_{Depth(K)}(K) = K$.
- Denote by $Ancestors_i(C)$ the collection of ancestor keys of the collection of keys C:

$$Ancestors_i(C) = \bigcup_{K \in C} Ancestor_i(K)$$

Note there is a one-to-one correspondence between nodes and keys:

$$Node(Key(N)) = N, \quad Key(Node(K)) = K$$

For simplicity, and since the original surface is a structured grid, the quadtree leaf nodes are assumed to be all at the same fixed depth. It is possible for branches of the quadtree to be empty. To ensure unnecessary navigation across empty parts of the tree, if a node is present in the tree it must have a non-empty leaf node in its descendants.

Conceptually, all surfaces are represented as mesh. However, it is not necessary for the mesh to be completely built, in fact, it is possible to dynamically build the mesh as

and when required provided sufficient topology is maintained to support geometrical and topological algorithms. Once an area of the surface is marked as mesh and simplices have been built for this region, more general, topological editing can be performed in this region, for example, refinement.

Following this paradigm, simplices are conceptually assigned to quadtree leaf nodes in a regular manner. As discussed above, this can be easily done for a grid. The quadtree leaf node maintains a flag that signifies whether its simplices have been built or not. If asked for its list of simplices the leaf node can return the list of simplices if they have been built, or build them dynamically and return the list. In this way a simplex is assigned to a unique quadtree leaf node.

The geometrical representation of a quadtree mirrors the regular structure of a grid and hence it is natural to assign each grid cell in a regular manner to a unique quadtree leaf node. For example, the geometric tiling of a quadtree described above can be used and the grid cell may be assigned to the quadtree leaf node that contains its lower left corner. When a grid cell is triangulated, it contains a pair of simplices. These simplices are assigned to the quadtree leaf node of their grid cell.

The quadtree leaf node that contains the simplex S will be denoted by Leaf(S).

Having made the assignment of simplices to quadtree leaf nodes, the quadtree hierarchy gives an equivalence relation for each depth i of the quadtree, by assigning simplices to their ancestor node at level i.

$$\textit{If } S \textit{ and } S' \textit{ are simplices then } S \sim S' \Rightarrow Ancestor_i(Leaf(S)) = Ancestor_i(Leaf(S'))$$

In other words, each quadtree node is assigned the simplices of its descendants and we define for a quadtree node:

$$Simps(N) = \{S | S \in N\}$$

As has been explained, each quadtree node has been assigned a collection of simplices. The quadtree node inherits a boundary from its collection of simplices, i.e., the boundary of the simplices. This boundary is a list of vertices and edges connecting the vertices. Furthermore, at a fixed depth in the tree, the boundaries of all the quadtree nodes at the chosen depth form a graph.

The graph of the boundaries of the quadtree node will be a particularly detailed object for it takes its edges from the simplices in the full-resolution surface. It would be preferable to approximate the boundary of the quadtree node with a reduced set of edges and hence a reduced set of vertices. For example, as shown in Fig. 24a, it can be seen that vertex 194 can be dropped and the edge 196 which it defines. The edge can be straightened, as shown in Fig. 24b, without changing the basic structure of the graph (more precisely, the topology of the graph has not changed).

In contrast, if vertex 198, shown in Fig. 25a, is collapsed to vertex 200, the valence of vertex 200 changes from three to four and hence the topology of the graph is modified. It can be seen that the only vertices which can be dropped and still preserve the topology of the graph are the vertices of valence two (such as vertex 194 in Fig. 24a). This can be made precise by introducing the notions of homeomorphic graphs. Graphs are well-known topological objects in mathematics and many techniques have been developed to study them.

From the description above, there is a collection of vertices that cannot be removed without changing the topology of the graph. These vertices are of interest as they encode the topology of the surface. However, it can be difficult to compute this collection of vertices. The following discussion describes a computationally cheap means to find a collection of vertices, called "critical vertices", which include the vertices described above.

The critical vertices are the crucial component when describing the topology of the surface. They enable navigation to be performed within the tree without requiring the complete tree to be loaded in memory. Moreover, they avoid creating unnecessary mesh

regions to describe the geometry of the surface. The navigation is generally an important part of a surface description but is essential in the operation of making coherent.

The critical vertices are also crucial in the decimation stage because they are the vertices that will appear in the decimated model.

There are two classes of critical vertices: "internal" critical vertices and "external" critical vertices. The internal critical vertices are present to provide topological connectivity in the interior of the surface. The external critical vertices provide topological connectivity around the boundary of the surface and along cracks in the surface.

Again, the driving example is the sub-sampling of a regular grid. The critical vertices mirror sub-sampling in a regular grid.

A vertex is an internal critical vertex if the vertex is in the interior of the surface and it can not be removed from the graph of edges without changing the topology of the graph. A vertex is critical at depth i if it is at the intersection of three or more quadtree nodes at depth i. For the regular partition of a grid, the critical vertices are the interior vertices of the sub-sampled grid, as can be seen in Fig. 26.

The internal critical vertices do not include the boundary vertices. To include these requires the macro-topology of the 2-cell to be used and gives the notion of an external critical vertex.

A vertex is an external critical vertex at depth i if it is:

- A vertex which is identified with a 0-cell.
- A vertex which is identified with a 1-cell vertex and lies at the boundary of two or more quadtree nodes at depth i.

The external critical vertices permit the boundaries to be included, as can be seen in Fig. 27.

The collection of critical vertices is the union of the internal critical vertices and the external critical vertices. One can see from the definitions if a vertex, v, is critical at depth d then the vertex is critical at all depths greater than d.

The "depth" of a critical vertex is the depth at which the vertex first becomes

critical. If a vertex is never critical in the tree, it will first appear in the vertices of the simplices of a leaf node and its depth is given by the depth of the tree plus 1. The critical vertices represent an approximation of the surface and at greater depths of the quadtree the approximation improves. Figure 28a shows an original surface with a hole comprising two 1-cells 198 and 200 and a crack comprising one 1-cell 202 with a quadtree overlaid. At this depth, there is only internal critical vertex 204 (occurring at the intersection of three of more quadtree nodes). There are external critical vertices 206, 208, 210, 212, 214, 216, 218, and 220 identified with a 0-cell. There are 9 external critical vertices 222, 224, 226, 228, 230, 232, 234, 236 and 238 identified with a 1-cell vertex of the surfaces and lying at the boundary of two or more quadtree nodes. Fig. 28b shows the approximation of the surface, hole and crack at this level of decimation. Curves between critical vertices are replaced by straight lines. For example, the portion of curve 198 between critical vertex 206 and critical vertex 228 is replaced by line 239.

Fig. 29a shows the same original surface overlaid with a quadtree at a depth one greater than that shown in Fig. 28a. The number of quadtree nodes quadruples from four to sixteen. Consequently, seven interior critical nodes 240, 242, 246, 248, 250, 252 and 254 are added. Further, 1-cell external critical vertices at the boundary of two or more quadtree nodes 256, 258, 260, 262, 264, 266, 268, 270, 272, 274, 276, 278, 280, 282, 284 and 286 are added. Fig. 29b shows the approximation of the surface, hole and crack at this level of decimation. Again, curves between critical vertices are replaced by straight lines.

As can be seen, assigning critical vertices to a quadtree node allows a decimated view of a surface to be built. For the quadtree node, N, let

$$CriticalVerts(N)=\left\{v \middle| v \in Verts(S) \text{ for some } S \in Simps(N) \text{ and } depth(v) \le depth(Key(N))\right\}$$

In its most abstract form, a 2-simplex is formed from three vertices. In a mesh, the connectivity of the surface can be represented by sharing the same vertex across

different simplices and maintaining within a vertex references to the simplices which reference the vertex. So, given a simplex, one can navigate to one of its vertices and from the vertex can navigate back to another simplex. In this manner one can travel around the simplices and vertices in a surface.

In the hierarchical surface representation, connectivity is built in a different manner. Within each critical vertex, instead of storing the simplices which reference the vertex, the keys of the quadtree leaf nodes which have the vertex as a critical vertex are stored. This list of keys is called the vertex descriptor. In this manner, it is not necessary to instantiate simplices to be able to navigate around the surface. Navigation can be performed from a quadtree node by using the quadtree leaf keys from the vertex descriptor of one of its critical vertices to navigate to an ancestor node of the key.

More precisely, each simplex belongs to a unique quadtree leaf node and the simplex is conceptually assigned the key of this leaf node. In the mesh representation of the surface, a vertex is shared by a list of simplices (these simplices may not be instantiated). Thus, the vertex inherits a list of leaf keys from the keys assigned to the simplices that reference the vertex. This list, which is the vertex descriptor, is defined for all vertices in the surface and for the critical vertices is the same list as the one described in the paragraph above.

It is not necessary to store the quadtree leaf key in a simplex because it can be computed from the vertex descriptors of the simplice's vertices, as described below. However, for efficiency, it can be cached at the simplex.

Within a cell, denote by Simps(v) the set of simplices that connect to the vertex v. Denote by Verts(S) the set of vertices forming the corners of the simplex S. The "vertex descriptor" for the vertex v, is the list of leaf keys of the simplices connected to the vertex, v,

$$Keys(v) = \bigcup_{S \in Simps(v)} Key(Leaf(S))$$

For performance, the tree is chosen such that the following condition is met:

$$Card(Keys(v)) \leq 4$$

By assigning the vertex descriptors to the vertices, it is possible to indirectly determine the leaf key of a simplex by

$$Key(Leaf(S)) = \bigcap_{v \in Verts(S)} Keys(v)$$

Having assigned the vertex descriptors to the vertices it is now a cheap operation to determine the criticality of a vertex at a particular depth of the tree.

Let *dim(v) = macro-dimension(v).* That is, let

$$\dim(v) = \begin{cases} 0, if\ v\ is\ identified\ with\ a\ 0-cell\ vertex \\ 1, if\ v\ is\ identified\ with\ a\ 1-cell\ vertex\ and\ no\ 0-cell\ vertex \\ 2, if\ v\ is\ not\ identified\ with\ a\ 1-cell\ or\ 0-cell\ vertex \end{cases}$$

Let $k_i(v)$ be the number of ancestor keys at depth i of the vertex v,

$$k_i(v) = Card(Ancestors_i(Keys(v))$$

The vertex v is critical at level i if it satisfies any of the following:

- The vertex v is identified with a 0-cell vertex.
- The vertex v is identified with a 1-cell vertex and the number of ancestor keys at depth i is greater than one.
- The number of ancestor keys at depth i is greater than two.

Equivalently, the vertex v is critical at level i if,

$$k_i(v) > \dim(v)$$

A quadtree node inherits a collection of simplices from the leaf nodes that are its descendants. If this collection of simplices does not reference any vertices that are identified with 0-cell or 1-cell vertices then, in the preferred embodiment, it is required that the collection of simplices must be homeomorphic to a 2-disk. In particular, the node must be connected and simply-connected (i.e. not have any holes).

This requirement improves the efficiency of the algorithms, in particular migration, and saves having to perform difficult topological analysis of the quadtree nodes. As will be seen, this requirement will be satisfied for all algorithms of interest provided the initial quadtree that is built satisfies the connectivity requirement.

The data structures of the quadtree has the following characteristics:

- Grid cells are assigned to quadtree leaf nodes in a regular manner, as described above.
- The initial triangulation of the surface is defined by splitting grid cells. The simplices from a split grid-cell are then assigned to the quadtree leaf node to which the grid-cell was assigned.
- Each quadtree leaf node maintains the list of simplices which it contains. If the simplices have not been built, this list is empty. The simplices are built dynamically from the grid cells when required. When the simplices have been built, the quadtree leaf node is marked as mesh and the grid representation can no longer be used.
- Each vertex is assigned a vertex descriptor that is the list of leaf keys of the simplices that use the vertex.
- Each quadtree node maintains the list of vertices that are critical for this quadtree node at the quadtree node's depth.
- Each quadtree node that has no vertices identified to 1-cell or 0-cell vertices in any of its descendants is topologically connected and simply connected.

A quadtree node is "pure grid" if none of its descendant leaf nodes have been meshed.

A "mesh" vertex is a vertex for which a simplex has been built which references this vertex.

A "grid" vertex is a vertex for which no simplices that reference it have been built.

A quadtree can be implemented in two basic forms. The first is to use a look up table based on the quadtree key, which can be done by using the ordering on the quadtree keys. The second is to maintain links from the parent to the four children nodes similar to a linked list. The first implementation will be called a linear array quadtree, the second implementation a linked quadtree.

A linear array quadtree is efficient for indexing on the key, but if large portions of the tree are non-existent it can be wasteful of memory. The linked quadtree is efficient where large portions of the tree are non-existent, but is inefficient for indexing on the key as it requires searching down the parent-child links.

The implementation of the quadtree uses both techniques. For the coarse depths the linear array is used and at the finer depths a linked tree is used. This provides faster indexing for coarse keys, but is not too wasteful of memory.

An iterator is provided for the tree, which maintains a key and provides efficient retrieval of the parent nodes. By using an iterator it is not necessary to maintain a key or the parent pointers in the quadtree nodes. They can be maintained in the iterator.

A bit tree is maintained which indicates whether a particular quadtree node is pure grid. This allows optimization on the basis that one knows whether a particular node is pure grid. For example, for a surface built from a regular grid, for a pure grid node, only the max z and min z values need be stored to recover a bounding box.

A bit vector for the grid cells is maintained which describes how a particular grid cell is split into simplices. This means it is not necessary to split all the grid cells to

define the simplicial structure.

For grid vertices, there is no need to build their vertex descriptor, because keys can be generated on the fly for nodes that are pure grid. So vertex descriptors are only needed for mesh vertices. Furthermore, one knows a quadtree node that is pure grid has four neighbors and hence a pure grid node can be migrated easily.

The implementation of the quadtree maintains the following data structures and implements the following algorithms.

- A bit vector, with one bit for each grid cell, to represent the triangulation of grid cells.
- A bit tree, with one bit for each node in the quadtree, to determine if a node is present in the tree. This is necessary to determine whether a node is present in persistent storage.
- A bit tree, with one bit for each node in the quadtree, to denote whether a node is pure grid or not. A node can be pure grid even though its boundary vertices may connect to mesh nodes. The bit tree is an efficient hierarchical encoding of the hybrid grid-mesh representation.
- An iterator is used to navigate in the tree. Thus the nodes do not need to store parent pointers or keys.
- Vertex descriptors are not needed for grid vertices.
- Simplices are built only when required, e.g. to support intersection curves.
- The quadtree key can be implemented in 32 bits.
- Coarse levels of the tree are implemented as a linear array quadtree. Finer levels are implemented as a linked quadtree.
- The mesh representation stored at the quadtree leaf nodes is the micro-topological representation defined in the SHAPES geometry engine. This is to save unnecessary conversion between different mesh

representations when passing geometry to the SHAPES geometry engine. If a geometry engine had an alternate mesh representation it would be possible to use this representation instead.

The implementation of the algorithms is now discussed, beginning with classification.

At each quadtree node a bounding box is maintained which is large enough to contain all the simplices assigned to that quadtree node.

To compute the intersection of two surfaces the bounding boxes can be used in a hierarchical manner to compute the quadtree leaf nodes that intersect. At this point, intersecting quadtree leaf nodes are resolved to their individual simplices that are then intersected to determine the intersection curve.

In the preferred embodiment, the underlying geometry engine computes the correct topology when all possible intersecting triangles are passed to the engine.

The next step is to make the model coherent. Making coherent consists of two fundamental steps. The first is to split simplices that lie along the macro-topological boundaries so the simplicial structure of the surface respects the micro-topology of its bounding 1-cells and 0-cells. The second is to collect the simplices into their respective topologically connected components, which is called migration.

In overview, new quadtrees are built for each new cell. As each simplex is split, each resulting simplex is assigned to the new quadtree for the cell. Using mesh connectivity, the remainder of the simplices which lie in the quadtree leaf nodes it belongs to are migrated. The quadtree nodes that are not split are then considered. These quadtree nodes are topologically connected and a flood fill using the vertex descriptors of the already migrated vertices is used to migrate these nodes. At this point, the critical vertices of each quadtree node in each new quadtree can be recomputed as described above. Finally, the bounding boxes are recomputed for each quadtree node.

A collection of nodes, C, of the quadtree T is a "node front" if every leaf node of

T either has no ancestor in C or has a unique ancestor in C (a node is an ancestor of itself).

A collection of nodes, C, of the quadtree T is a "complete node front" if C is a node front and every leaf node of T has an ancestor in C.

A subtree S of T can be built from the node front C, by taking all of the ancestors of C in T. The tree structure of S is inherited from T and the collection C forms the leaves of S.

The process begins with the surface being partitioned into n_i nodes at resolution level-i using an i-level subset of boundaries 288, as shown in Fig. 30a. Each level-i+1 node is associated with a unique level-i node 290. Each level-i node is associated with the level-i+1 nodes associated to the node 292. Each level-i node has associated with it a subset of the vertices that are critical at resolution level i 294. Each node at resolution level d is designated a leaf node 296. While the preferred embodiment has leaf nodes at the same level, it is also possible to have leaf nodes at numerous levels of resolution.

Each simplex is associated with a unique leaf node 298. Each leaf node has associated with it the simplices associated to that leaf node 300. Each level-i node has associated with it the list of simplices which is the union of all simplices associated with the level-i+1 nodes grouped under the level-i node 302. The subset of boundaries for each node is assigned to be the boundary of the union of the simplices associated with that node 304. Each node is assigned a unique key and each vertex in a leaf node is assigned a key corresponding to that leaf node 308.

Now assume that a second surface is classified into the model. It is determined which leaf nodes of the first surface intersect the leaf nodes of the second surface 310, as shown in Fig.30b. The intersecting simplices from the first and second surfaces are determined from the simplices associated to the intersecting leaf nodes 312. The original quadtree is split into new quadtrees and each new quadtree is associated with a new cell 314. The subtrees of the original quadtree which have no intersecting leaf nodes are identified with one of the new cells 316.

The simplices of the first surface are split along the intersection curve 318, as shown in Fig. 30c. New simplices are formed by tessellating the split simplices to respect the macro-topology of one-cells and zero-cells passing through the original simplices 320. A new quadtree is built for each new cell 322. Each new simplex is assigned to the leaf node of the quadtree created for the new cell to which the new simplex belongs 324. For each leaf node of each new quadtree, each simplex in the original quadtree which is connected to a new simplex in the new quadtree leaf node and which lies in the same quadtree leaf node as the new simplex is migrated 326. The neighbors of a quadtree node are determined by finding all the keys of the critical vertices in the node 328. The coarsest level node which is an ancestor of a key from the critical vertices in the migrated quadtree nodes and which has not been split or migrated is determined and migrated to the new quadtree 330.

Decimation begins with a list of critical vertices being built from the quadtree nodes of a complete node front 332, as shown in Fig. 30d.

Figure 31 is an example of a quadtree with leaf nodes at a fixed depth. Examples of node fronts are {b,c,q,s,t} and {a,c,d}. Examples of complete node fronts are {a,b,c,d} and {a,i,j,k,l,c,d}. An example of a non-node front is {a,e,b,c,d}. The collection {a,e,b,c,d} is not a node front because "e" does not have a unique ancestor in the collection.

The vertices identified to one- or zero-cell vertices is removed from the list 334 (Fig. 30d). All zero-cell vertices from the model which lie in the first surface are added to the list 336. A defined collection of one-cell vertices is added to the list 338. The collection of one-cell edges is recorded 340. The surface is tessellated to respect the list of vertices and the recorded one-cell edges 342. A requirement is imposed that the subset of vertices on the boundary of the first surface which are also on the boundary of the second surface be the same as the subset of vertices on the boundary of the second surface which are also on the boundary of the first surface 344.

A first surface is partitioned into n_i nodes at resolution level-i using an i-level set

of boundaries 288 (Fig. 30a). A geometrical representation of the first surface is maintained in persistent storage 346 (Fig. 30e). For each node, a bounding box is stored on a persistent storage device 348. For each node, a list of critical vertices associated with that node is stored on the persistent storage device 350. For each critical vertex, a vertex descriptor, a parameter value and an image value are stored on the persistent storage device 352. The required portion of the first surface is loaded on demand from the persistent storage device 354 and that portion of the first surface not required is removed 356. A quadtree node is loaded on demand from persistent storage and removed when it is no longer needed 358. A quadtree leaf node is loaded on demand from persistent storage and removed when it is no longer needed 360.

In this way memory usage is conserved, and furthermore changes to the model are limited to the particular collection of sub-volumes specified.

Even with the ability to selectively load geometry at the macro-topological level there is still a memory usage problem. For example, the user of the GQI may want to load the whole earth model and view it. Using the multi-resolution hierarchy it is possible to selectively load portions of the geometry of the surfaces. For a user who is viewing the whole earth model it is not necessary to load the fine details of the model. It is sufficient to load a collection of nodes from the coarse levels of the quadtrees to give a good approximation to the earth model. For a user who is viewing a small part of the whole earth model, it is not necessary to load the finer levels of the quadtrees not in the viewing volume.

Consider computing the intersection of two surfaces. This requires loading all pairwise intersecting leaf nodes and then loading the simplices which these leaf nodes contain. This can be done in a recursive manner as follows. Load the bounding box for the root node of each quadtree of each surface. If there is no intersection between the bounding boxes then the two surfaces do not intersect. If there is an intersection choose one of the nodes and load its children's bounding boxes and mark the node's bounding box for

removal from memory. Now intersect the children's bounding boxes with the bounding box of the node from the other tree. If there is an intersection then recurse down the branches of the trees by loading the bounding boxes of the children nodes and intersecting with the bounding boxes from the other tree until the leaf nodes are reached. If there is no intersection then stop the recursion. This results in pairs of intersecting quadtree leaf nodes. Now load all the simplices which are contained in the leaf nodes and pass them to the geometry engine.

The partial loading algorithm described above for the intersection algorithm can be applied to all algorithms which reference information in the hierarchical surface. In particular, it can be applied to the migration algorithm where nodes, simplices and vertices can be selectively loaded from disk.

To achieve optimum performance, it is essential, when local changes are made to the model, that it is possible to map these changes to local updates in the persistent storage. An example of a local change is to modify the (x,y,z) position of a vertex. To maintain efficient persistent storage, it is essential this (x,y,z) position is at a limited number of locations in persistent storage, and preferably should be at a unique location. The mapping architecture is described below and is achieved by mapping the quadtree structure to persistent storage.

A quadtree node consists of a fixed size component, the bounding box, and a variable sized component, the list of critical vertices. For efficiency, these two components are maintained in separate locations in persistent storage. In persistent storage the fixed size component models the situation in memory and uses a mix of linear array indexing with linked-list indexing at the finer levels.

The architecture for storing the critical vertices is more complicated and is described below.

When saving a vertex to disk it is necessary to store the vertex descriptor (the list of leaf keys), the parameter value of the vertex, and the image value of the vertex.

All vertices that are identified with 0-cell or 1-cell vertices are stored separately and are loaded whenever the surface is loaded into core memory. The SHAPES geometry engine maintains an index that is used to refer to the 0-cell or 1-cell vertices this vertex is identified with. For the geometry engine to rebuild the identification structures the vertex must store its index and on loading must restore the index.

All other critical vertices are stored at the quadtree node, which is the finest common ancestor of all the leaf keys in the vertex descriptor.

A quadtree node has three types of critical vertices, illustrated in Fig. 32. The type of critical vertex can be determined from the number of unique keys at the parent level.

Let v be a critical vertex at level i+1. As discussed above, this implies:

$$k_{i+1}(v) > \dim(v)$$

Accordingly, v must be either parent critical, edge critical or sub-critical:

1. v is "parent critical"(e.g. node 362, Fig. 32) if:

$$k_i(v) > \dim(v)$$

2. v is "edge critical" (e.g. nodes 364 and 366) if the vertex was not critical at the parent level and lies on the boundary of the parent node:

$$k_i(v) = \dim(v)$$

3. v is "sub-critical" (e.g. node 368) if the vertex is neither parent critical nor edge critical:

$$k_i(v) < \dim(v)$$

The critical vertices of a quadtree node are maintained in three lists, the parent critical, the edge critical and the sub-critical vertices. For the parent critical vertices the list is a list of indices into the parent critical vertices. For the sub-critical vertices, the vertex is stored at the parent node, because the parent is the finest common ancestor of the list of leaf keys. The list of sub-critical vertices is a list of indices into the vertices stored at the parent node. Identifying an edge critical vertex requires a quadtree key, which specifies the node, which stores the vertex, together with an index into the list of stored vertices of the node

that stores the vertex.

The list of vertices which is stored at a particular node can be broken into a collection of buckets. For example, every vertex has a unique depth, this being the depth when the vertex first becomes critical. For the quadtree nodes that are at the vertex's depth and have the vertex as a critical vertex, the vertex must be either edge critical or sub-critical for all these quadtree nodes. Thus, the list of stored vertices can be broken into vertices that will be edge critical vertices and vertices that will be sub-critical vertices. The edge critical vertices can be broken down further by finding the pair of quadtree keys at the depth of the vertex and by identifying the child index of each key in its parent which gives a pair of two bit indices which when ordered can be used to identify a bucket. The edge bucket can be further broken down by using the depth of each vertex. In fact, an edge bucket inherits a binary tree from the quadtree structure (it is the restriction of the quadtree to the particular edge), and this can be used to provide hierarchical loading of the edge critical vertices.

The quadtree leaf node has an additional component beyond the other nodes in the tree and that is the mesh representation of the node. This is a simple list of triples of indices specifying the indices into the critical vertices, edge vertices and internal vertices of this quadtree node. As illustrated in Fig. 33, a grid representation of a surface is stored, the grid being made up of grid cells 370. A mesh representation of a portion of the surface is then formed by triangulating a subset of the grid cells 372.

Since the quadtree leaf node must contain the list of simplices, the mesh content of the leaves is stored in a separate location on the disk. This storage is implemented using a blocked linked list. This allows maximum flexibility for adding and deleting simplices from the database, but provides an efficient means to load a particular leaf node.

It is also possible to implement a multiresolution hierarchy for multivalued surfaces, as discussed below.

The GQI must be able to import a triangulated surface defined as an Inventor face set. The simplices in the face set can be oriented in a completely arbitrary manner, and

overall the surface can be multi-valued and non-manifold. While it is possible to formally apply the quadtree partitioning scheme to such a surface, it is unlikely that the tiling element patches will be connected. What is needed is a mapping of the surface onto a planar region that preserves the surface's simplicial connectivity.

One approach for constructing this mapping is to model the surface deformation as an energy minimization problem whose solution is consistent with Hamilton's Least Action Principle:

> Given an elastic surface g(u,v,t) = (x(u,v,t), y(u,v,t), z(u,v,t)) that is to be deformed into a shape g'(u,v,t) = (x'(u,v,t), y'(u,v,t), z'(u,v,t)). Assume that a set of imaginary massless springs join sample points on g to their final place on g'. Then the motion of deformation follows a path in time such that the action
>
> $$\int_a^b (K(g,g',u,v,t) - P(g,g',u,v,t))dt$$
>
> is minimized. (The time interval is [a,b], K is the surface system's kinetic energy, and P is its potential energy. The difference K-P is the system's "LaGrangian".)

In order to apply this principle to triangulated surfaces, the surface must be "sampled" at its vertices. An application can constrain the potential energy in time three ways:

1. By the position of each vertex. The position constraint applied to the potential energy stored in the springs is defined by

   $$\frac{1}{2}\iint C(g'-g)^2 dudv$$

   where C is a spring constant that is a function of the (u,v) sample coordinate. Frequently, C is independent of the (u,v) sample coordinate. The Euler-LaGrange theorem applied to the minimization of this integral says that the deformation g'(u,v,t) must converge to g(u,v,t) over

time.

2. By the orientation of the surface at each sample vertex. The orientation constraint is defined by

$$\frac{1}{2}\iint D((\frac{\partial g'}{\partial u}-\frac{\partial g}{\partial u})^2+(\frac{\partial g'}{\partial v}-\frac{\partial g}{\partial v})^2)dudv$$

where D denotes a different spring constant. Again, D is usually set to a fixed common value. The Euler-LaGrange theorem applied to the minimizing function for this integral says that the deformation g(u,v,t) must be a harmonic function. This is good, because a harmonic function causes the least distortion to the aspect ratio of the simplices on the deformed surface. A sliver results when the aspect ratio of a simplex is poor. Therefore, if the intention is to tessellate the planar deformation, then minimizing the number of slivers and their degree of malformation is important.

3. Finally, the curvature constraint applied to the spring's potential energy is given by a minimizing function for the integral equation,

$$\frac{1}{2}\iint E((\frac{\partial^2 g'}{\partial u^2}-\frac{\partial^2 g}{\partial u^2})^2+(\frac{\partial^2 g'}{\partial u\partial v}-\frac{\partial^2 g}{\partial u\partial v})^2+(\frac{\partial^2 g'}{\partial v^2}-\frac{\partial^2 g}{\partial v^2})^2)dudv$$

where E is a spring constant. Again, E is usually set to a fixed common value. The Euler-LaGrange theorem says that a minimizing function for this constraint satisfies a biharmonic relation. This form of constraint is used in some grid-based surface modeling systems. Typically, the full surface is subdivided into small patches. On each patch a solution is computed, and some form of smoothing is applied across the interface between two patches.

A GQI surface is triangulated, so the minimizing function must preserve the valence of every vertex. Hence a linear harmonic deformation is ideal. Hamilton's Principle says nothing about how vertex pairs, i.e., the edges that form the topology,

deform. This is significant, because a smooth vertex deformation can reposition the vertex endpoints of a set of connected simplex edges so that the deformed edges cross, failing to preserve the surface's topological definition. When this happens, the GQI has to split the surface before the folding occurs and restart the process on the remainder of the surface. Thus, the deformation of a surface feature may be defined as a large number of cells. A large cell count slows down classification, and forces the GQI to create a quadtree for each 2-cell, exploding memory. What is needed is a method to parameterize the surface given just the surface's vertex and simplex connectivity, i.e., a topology based method.

A method is described below for construction of a single 2-cell for the case that the surface is oriented with an arbitrary number of holes. This method does not control metric distortion, in contrast to a harmonic deformation. The GQI computes this deformation in order to construct a quadtree rather than as preparation for further tessellation, so some distortion is acceptable. When applied to a gridded triangulated surface, the mapping is equivalent to the projection of the surface onto one of the coordinate planes.

Let F, E, and V be the number of faces, edges, and vertices, respectively, of a triangulated surface. The Euler Characteristic X of the surface is defined as $X = F - E + V$. When the surface is presented in "normal" form, the Euler Characteristic X of an oriented surface is equal to $X = 2 - (2 \cdot g + r)$, where g denotes the genus of the surface (which equals the number of embedded tori) and r is the number of boundary curves ("holes"), grouped into GQI frames. Fig. 34 lists some common surfaces and their Euler Characteristics.

A standard result in topology is that two oriented surfaces are homeomorphic if and only if they have the same genus and number of frames. The Euler Characteristic is a function of genus and the number of frames, so it follows that the Euler Characteristic is independent of the triangulation. The GQI supports queries for the evaluation of F, E, and V, as well as frames. Thus two oriented triangulated surfaces that have the same Euler

Characteristic and the same number of frames must be homeomorphic.

In geological applications, multivalued surfaces such as a recumbent fault, a salt body, or a horizontal wellbore are encountered. All three surfaces have genus zero and have Euler Characteristic 0, 1 or 2, assuming that the surface has zero internal holes. The quadtree construction assumes that the surface is single-valued in its (u,v) coordinates, so to apply the quadtree construction to a multi-valued surface, it is necessary to construct a homeomorphism between the surface and a single-valued representation with the correct Euler Characteristic. If the original surface has Euler Characteristic 1, e.g., a recumbent fault, a good choice is to map it onto a planar square or family of connected squares. If the original surface has Euler Characteristic 0, e.g., a wellbore, then the square should be replaced by a squared annulus. Finally if the surface has Euler Characteristic 2, then it can be split into two equal-area parts with each part of Euler Characteristic 1 with the previous construction applied to each part.

Here are the steps in the construction of the homeomorphism between a set of connected planar squares and an oriented genus 0 triangulated surface with Euler Characteristic

1. Build the surface as a non-parametric web, so that the non-manifold boundaries are apparent. From now on, assume that the surface is a 2-manifold.
2. Find the surface's external boundary curve, which is defined by the zeroth GQI frame instance. Analyze the external boundary for embedded disc obstructions, as shown in Fig. 35. Remove each Type (a) obstruction by adding a new vertex plus edges to form a new simplex that uses the obstructing vertex as a corner, as shown in Fig. 36. If a Type (b) obstruction is detected, then exclude it from the remainder of the algorithm and at the conclusion add it to the final planar square. If a Type (c) obstruction is detected, then split the surface into three parts,

building two squares and a connector, essentially reducing to a Type (b) problem.

3. Compute the area of the (possibly extended) surface, ignoring Type (b) obstructions, and construct a square of that size. The area of each surface simplex can be computed from the cross product of two sides of the simplex, thought of as vectors. Randomly select four points along the external boundary that are spaced roughly 1/4 of the contour apart from each other. These points are mapped to the corners of the square. The remaining points are assigned in relative proportion to the appropriate side of the square. Initialization is now complete.

4. Determine the contour formed from the set of all vertices connected to the most recently defined contour, as shown in Fig. 37. First find four vertices on the new contour that are connected to the corners of the old contour. If no edge exists between a corner to the new contour, then split the obstructing simplex. Note that this might cause other simplices to split. A good choice of a second vertex is the simplex's barycenter. In part (a) of Fig. 37, the dotted edge has been added to join the two corners.

5. If two edges of the contour form a Type (a) obstruction, then redefine the new contour to include that simplex unless the obstruction simplex's valence 2 vertex is directly connected to a corner vertex of the previous contour. In part (b) of Fig. 37, the dark simplex is a Type (a) obstruction. Assuming that the valence 2 vertex is not directly connected to a corner vertex of the previous contour, remove the obstruction by incorporating the obstructing simplex's remaining edge into the new contour, effectively flattening the contour. In Fig. 37 the two offending interior corners do not obstruct the direct connection to

corner vertices of the previous contour, so both offenders can be safely ignored.

6. Since the full surface is oriented and of genus 0, the set of simplices connecting the two contours is also oriented and of genus 0. Assuming that the surface has no holes and contains no Type (a) obstructions and no non-manifold linkage of the interior and exterior contours, it follows that the surface patch defined by this set of simplices has Euler Characteristic. Referring to Fig. 38, the patch is homeomorphic to the lateral surface of a truncated square-based pyramid. Deform the surface formed by the simplices between the two contours into the lateral surface of a truncated square-based pyramid, then project the upper contour onto the interior of the square defined by the lower contour, as shown in Fig. 38. Note that two edges on a lateral face cross if and only if their projection cross, explaining why the choice of a lateral surface of a square-based pyramid. If a corner simplex was split in step #4, then replace its split parts by the projection of the original simplex.

7. It is possible that the interior and exterior contours together describe a connected sum of lateral surfaces of square-based pyramids, due to Type (a) obstructions, or meet in a non-manifold manner, see Fig. 39. An example of this would be an otherwise flat surface on which two hills exist. If this happens, the process divides the surface into the number of connected components identified by the interior contour and applies the construction to each component. (Note that the hexagonal region between the two shaded areas has Euler Characteristic 1, so it is homeomorphic to a disc.) When this happens, the algorithm is applied to each sub-square and any Type (a) obstructions are filled in. A homeomorphism preserves continuity, so the number of embedded

squares equals the number of hills and valleys in the multi-valued surface. Note that running the construction in reverse is a simple way to create a surface with any number of hills and valleys, salt domes, etc.

8. Repeat this identification until the current contour is the boundary of an embedded disk, as shown in Fig. 40. Complete the square construction by deforming the indicated disk to a square and adding it in.

9. It is possible that there may not be four distinct vertices on a contour, e.g., a pyramid formed with a triangle cap and a square base. If this happens, then the nested square is replaced by a triangle and the construction proceeds as above, as shown in Fig. 41.

Loosely speaking, the homeomorphism "melts" the surface onto a plane. Consequently, points close together in (x,y,z) space may not be close together in the plane. (This is another way of saying that this method does not control metric distortion.) An extreme example is a spherical surface with a tiny hole at a pole. If the hole is used as the initial boundary, then the triangles forming the boundary can be quite distorted in the mapping plane. When the surface is single-valued, it may be acceptable to use the projection of the bounding box onto the appropriate plane as the initial square and create nested square-sided annuli that proportional with respect to area. In any case, distortion is not a problem, since the planar version of the surface is used only to facilitate the quadtree decomposition of the surface. The GQI constructs the quadtree on the homeomorphic planar region, then lifts it back. The only significant change in the quadtree hierarchy API is that ray picking can return more than one tiling element even when the ray does not pick a simplex corner.

This construction can be applied to a surface with an arbitrary number of holes in it by adding to the area of the tiling cover of each hole to the area of the original surface. Also, this approach can tile a toroidal shape. Indeed, all that is needed is to randomly choose two closed paths in opposing directions, then "unroll" the toroidal shape along the

two paths. Assuming that the surface has no holes, the two formulas for the Euler Characteristic can be equated to infer when a toroidal shape is present and the quadtree construction applied to each toroidal element. Not much is made of this, however, because toroidal structures are seldom encountered in geological modeling.

The invention has application outside the field of geological modeling. In particular, the invention has application in any field in which data are presented geometrically and graphically at more than one level of resolution. For example, the invention would be useful in representing medical models, such as those developed in magnetic resonance imaging ("MRI"). Further, the invention would be useful in representing computer aided design ("CAD") models.

The invention may be implemented in hardware or software, or a combination of both. However, preferably, the invention is implemented in computer programs executing on programmable computers each comprising a processor, a data storage system (including volatile and non-volatile memory and/or storage elements), at least one input device, and at least one output device. Program code is applied to input data to perform the functions described above and generate output information. The output information is applied to one or more output devices, in known fashion.

Each program is preferably implemented in a high level procedural or object oriented programming language (such as C++ or C) to communicate with a computer system. However, the programs can be implemented in assembly or machine language, if desired. In any case, the language may be a compiled or an interpreted language.

Each such computer program is preferably stored on a storage media or device (*e.g.*, ROM or magnetic/optical disk or diskette) readable by a general or special purpose programmable computer, for configuring and operating the computer when the storage media or device is read by the computer to perform the procedures described herein. The inventive system may also be considered to be implemented as a computer-readable storage medium, configured with a computer program, where the storage medium so configured

causes a computer to operate in a specific and predefined manner to perform the functions described herein.

Other embodiments are within the scope of the following claims.

What is claimed is:

1. A method for representing a first surface at multiple levels of resolution, the first surface comprising zero or more zero-cells, zero or more one-cells and one or more two cells, the method being implemented in a programmed computer comprising a processor, a memory, a persistent storage system, at least one input device, and at least one output device, the method and a model being stored on a computer-readable media, the method representing the model on one of the output devices, the method comprising:

partitioning the first surface with one or more boundaries, each level of resolution having a subset of the boundaries.

2. The method of claim 1 further comprising:

partitioning the first surface into n_i nodes at resolution level-i using the level-i subset of boundaries;

associating each level-i+1 node with a unique level-i node;

associating with each level-i node the level-i+1 nodes associated to the node.

3. The method of claim 2 further comprising:

associating with each level-i node a subset of vertices that are critical at resolution level i.

4. The method of claim 3, wherein level d is the deepest level of resolution and wherein the first surface is divided into simplices, further comprising:

designating each node at resolution level d a leaf node;

associating each simplex to a unique leaf node; and

associating with each leaf node the simplices associated to that leaf node.

5. The method of claim 4 further comprising:

associating with a level-i node the list of simplices which is the union of all simplices associated with the level-i+1 nodes associated to the level-i node.

6. The method of claim 5 further comprising assigning the subset of boundaries for each node the boundary of the union of the simplices associated with that node.

7. The method of claim 6 wherein the nodes form an original tree and each node is assigned a unique key.

8. The method of claim 7 further comprising:

assigning to each vertex in a leaf node the key corresponding to that leaf node.

9. The method of claim 8 further comprising:

storing the representation of a second surface in the computer-readable media, the second surface having nodes, leaf nodes, vertices, critical vertices and simplices.

10. The method of claim 9 further comprising:

determining which leaf nodes of the first surface intersect the leaf nodes of the second surface;

determining the intersecting simplices from the first and second surfaces from the simplices associated to the intersecting leaf nodes.

11. The method of claim 4 wherein each node except the leaf nodes has a subtree, further comprising splitting the original tree into new trees and associating each new tree with a new cell.

12. The method of claim 11 further comprising identifying the subtrees of the original tree which have no intersecting leaf nodes with one of the new cells.

13. The method of claim 12 further comprising:

splitting the simplices of the first surface along the intersection curve;

forming new simplices by tessellating the split simplices to respect the macro-topology of one-cells and zero-cells passing through the original simplices;

building a new tree for each new cell;

assigning each new simplex to the leaf node of the tree created for the new cell to the which the new simplex belongs;

for each leaf node of each new tree, migrating each simplex in the original tree which is connected to a new simplex in the new tree leaf node and which lies in the same tree leaf node as the new simplex;

determining the neighbors of a tree node by finding all the keys of all the critical vertices in the node; and

determining the coarsest level node which is an ancestor of a key from the critical vertices in the migrated tree nodes and has not been split or migrated and migrating that node to the new tree.

14. The method of claim 8, wherein a complete node front of the tree of the first surface and a collection of vertices on a boundary of the first surface are defined, further comprising:

building a list of critical vertices from the tree nodes of the complete node front;

removing from the list those vertices identified to one- or zero-cell vertices;

adding to the list all zero-cell vertices from the model which lie in the first surface;

adding to the list the defined collection of one-cell vertices;

recording the collection of one-cell edges; and

tessellating the surface to respect the list of vertices and the recorded one-cell edges.

15. The method of claim 14 further comprising:

requiring the subset of vertices on the boundary of the first surface which are also on the boundary of the second surface to be the same as the subset of vertices on the boundary of the second surface which are also on the boundary of the first surface.

16. The method of claim 1 further comprising:

maintaining in a persistent storage a geometrical representation of the first surface.

17. The method of claim 2 further comprising:

storing for each node a bounding box on a persistent storage device, and storing for each node a list of critical vertices associated with that node on the persistent storage device.

18. The method of claim 17 wherein storing the list of critical vertices comprises:

storing a vertex descriptor for each vertex;

storing a parameter value for each vertex; and

storing an image value for each vertex.

19. The method of claim 18 further comprising loading on demand from persistent storage into memory that portion of the first surface required and removing from memory that portion of the first surface not required.

20. The method of claim 19 wherein a tree node is loaded from persistent storage on demand and is removed from memory when no longer required.

21. The method of claim 20 wherein the simplices associated with a tree leaf node are loaded from persistent storage on demand and removed from memory when no longer required.

22. A method for representing a first surface at multiple levels of resolution, the method being implemented in a programmed computer comprising a processor, a data storage system, at least one input device, and at least one output device, the method and a model being stored on a computer-readable media, the method representing the model on one of the output devices, the method comprising:

storing a grid representation of the first surface, the grid representation being made up of grid cells;

forming a mesh representation of a portion of the first surface by triangulating a subset of the grid cells; and

inserting the first surface into the model.

23. A computer system for representing a first surface at multiple levels of resolution, the first surface comprising zero or more zero-cells, zero or more one-cells and one or more two cells, the computer system comprising a processor, a data storage system, at least one input device, and at least one output device, the first surface being stored on the data storage system, the computer system comprising:

means for partitioning the first surface with one or more boundaries, each level of resolution having a subset of the boundaries.

24. The computer system of claim 23 further comprising:

means for partitioning the first surface into n_i nodes at resolution level-i using the level-i subset of boundaries;

means for associating each level-i+1 node with a unique level-i node;

means for associating with each level-i node the level-i+1 nodes associated to the node.

25. The computer system of claim 24 further comprising:

means for associating with each level-i node a subset of vertices that are critical at resolution level i.

26. The computer system of claim 25, wherein level d is the deepest level of resolution and wherein the first surface is divided into simplices, further comprising:

means for designating each node at resolution level d a leaf node;

means for associating each simplex to a unique leaf node; and

means for associating with each leaf node the simplices associated to that leaf node.

27. The computer system of claim 26 further comprising:

means for associating with a level-i node the list of simplices which is the union of all simplices associated with the level-i+1 nodes associated to the level-i node.

28. The computer system of claim 27 further comprising means for assigning the subset of boundaries for each node the boundary of the union of the simplices associated with that node.

29. The computer system of claim 28 wherein the nodes form an original tree and each node is assigned a unique key.

30. The computer system of claim 29 further comprising:

means for assigning to each vertex in a leaf node the key corresponding to that leaf node.

31. The computer system of claim 30 further comprising:

means for storing the representation of a second surface in the computer-readable media, the second surface having nodes, leaf nodes, vertices, critical vertices and simplices.

32. The computer system of claim 31 further comprising:

means for determining which leaf nodes of the first surface intersect the leaf nodes of the second surface;

means for determining the intersecting simplices from the first and second surfaces from the simplices associated to the intersecting leaf nodes.

33. The computer system of claim 26 wherein each node except the leaf nodes has a subtree, further comprising means for splitting the original tree into new trees and associating each new tree with a new cell.

34. The computer system of claim 33 further comprising means for identifying the subtrees of the original tree which have no intersecting leaf nodes with one of the new cells.

35. The computer system of claim 34 further comprising:

means for splitting the simplices of the first surface along the intersection curve;

means for forming new simplices by tessellating the split simplices to respect the macro-topology of one-cells and zero-cells passing through the original simplices;

means for building a new tree for each new cell;

means for assigning each new simplex to the leaf node of the tree created for the new cell to the which the new simplex belongs;

for each leaf node of each new tree, means for migrating each simplex in the original tree which is connected to a new simplex in the new tree leaf node and which lies in the same tree leaf node as the new simplex;

means for determining the neighbors of a tree node by finding all the keys of all the critical vertices in the node; and

means for determining the coarsest level node which is an ancestor of a key from the critical vertices in the migrated tree nodes and has not been split or migrated and migrating that node to the new tree.

36. The computer system of claim 30, wherein a complete node front of the tree of the first surface and a collection of vertices on a boundary of the first surface are defined, further comprising:

means for building a list of critical vertices from the tree nodes of the complete node front;

means for removing from the list those vertices identified to one- or zero-cell vertices;

means for adding to the list all zero-cell vertices from the model which lie in the first surface;

means for adding to the list the defined collection of one-cell vertices;

means for recording the collection of one-cell edges; and

means for tessellating the surface to respect the list of vertices and the recorded one-cell edges.

37. The computer system of claim 36 further comprising:

means for requiring the subset of vertices on the boundary of the first surface which are also on the boundary of the second surface to be the same as the subset of vertices on the boundary of the second surface which are also on the boundary of the first surface.

38. The computer system of claim 23 further comprising:

means for maintaining in a persistent storage a geometrical representation of the first surface.

39. The computer system of claim 24 further comprising:

means for storing for each node a bounding box on a persistent storage device, and storing for each node a list of critical vertices associated with that node on the persistent storage device.

40. The computer system of claim 39 wherein the means for storing the list of critical vertices comprises:

means for storing a vertex descriptor for each vertex,

means for storing a parameter value for each vertex, and

means for storing an image value for each vertex.

41. The computer system of claim 40 further comprising means for loading on demand from persistent storage into memory that portion of the first surface required and means for removing from memory that portion of the first surface not required.

42. The computer system of claim 41 wherein the means for loading loads a tree node from persistent storage on demand and the means for removing removes a tree node from memory when it is no longer required.

43. The computer system of claim 42 wherein the means for loading loads simplices associated with a tree leaf node from persistent storage on demand and the means for removing removes simplices from memory when they are no longer required.

44. A computer system for representing a first surface at multiple levels of resolution, the computer system comprising a processor, a data storage system, at least one input device, and at least one output device, the first surface being stored on the data storage system, the computer system comprising:

means for storing a grid representation of the first surface, the grid representation being made up of grid cells;

means for forming a mesh representation of a portion of the first surface by triangulating a subset of the grid cells; and

means for inserting the first surface into the model.

45. A computer program, residing on a computer-readable medium, comprising instructions for causing a computer, comprising a processor, a data storage system, at least one input device, and at least one output device, to represent a first surface at multiple levels of resolution, the first surface comprising zero or more zero-cells, zero or more one-cells and one or more two cells, the computer program comprising instructions for causing the computer to:

partition the first surface with one or more boundaries, each level of resolution having a subset of the boundaries.

46. The computer program of claim 45 further comprising instructions for causing the computer to:

partition the first surface into n_i nodes at resolution level-i using the level-i subset of boundaries;

associate each level-i+1 node with a unique level-i node;

associate with each level-i node the level-i+1 nodes associated to the node.

47. The computer program of claim 46 further comprising instructions for causing the computer to:

associate with each level-i node a subset of vertices that are critical at resolution level i.

48. The computer program of claim 47, wherein level d is the deepest level of resolution and wherein the first surface is divided into simplices, further comprising instructions for causing the computer to:

designate each node at resolution level d a leaf node;

associate each simplex to a unique leaf node; and

associate with each leaf node the simplices associated to that leaf node.

49. The computer program of claim 48 further comprising instructions for causing the computer to:

associate with a level-i node the list of simplices which is the union of all simplices associated with the level-i+1 nodes associated to the level-i node.

50. The computer program of claim 49 further comprising instructions for causing the computer to assign the subset of boundaries for each node the boundary of the union of the simplices associated with that node.

51. The computer program of claim 50 wherein the nodes form an original tree and each node is assigned a unique key.

52. The computer program of claim 51 further comprising instructions for causing the computer to:

assign to each vertex in a leaf node the key corresponding to that leaf node.

53. The computer program of claim 52 further comprising instructions for causing the computer to:

store the representation of a second surface in the computer-readable media, the second surface having nodes, leaf nodes, vertices, critical vertices and simplices.

54. The computer program of claim 53 further comprising instructions for causing the computer to:

determine which leaf nodes of the first surface intersect the leaf nodes of the second surface; and

determine the intersecting simplices from the first and second surfaces from the simplices associated to the intersecting leaf nodes.

55. The computer program of claim 48 wherein each node except the leaf nodes has a subtree, further comprising instructions for causing the computer to split the original tree into new trees and to associate each new tree with a new cell.

56. The computer program of claim 55 further comprising instructions for causing the computer to identify the subtrees of the original tree which have no intersecting leaf nodes with one of the new cells.

57. The computer program of claim 56 further comprising instructions for causing the computer to:

split the simplices of the first surface along the intersection curve;

form new simplices by tessellating the split simplices to respect the macro-topology of one-cells and zero-cells passing through the original simplices;

build a new tree for each new cell;

assign each new simplex to the leaf node of the tree created for the new cell to the which the new simplex belongs;

for each leaf node of each new tree, migrate each simplex in the original tree which is connected to a new simplex in the new tree leaf node and which lies in the same tree leaf node as the new simplex;

determine the neighbors of a tree node by finding all the keys of all the critical vertices in the node; and

determine the coarsest level node which is an ancestor of a key from the critical vertices in the migrated tree nodes and has not been split or migrated and migrate that node to the new tree.

58. The computer program of claim 52, wherein a complete node front of the tree of the first surface and a collection of vertices on a boundary of the first surface are defined, further comprising instructions for causing the computer to:

build a list of critical vertices from the tree nodes of the complete node front;

remove from the list those vertices identified to one- or zero-cell vertices;

add to the list all zero-cell vertices from the model which lie in the first surface;

add to the list the defined collection of one-cell vertices;

record the collection of one-cell edges; and

tessellate the surface to respect the list of vertices and the recorded one-cell edges.

59. The computer program of claim 58 further comprising instructions for causing the computer to:

require the subset of vertices on the boundary of the first surface which are also on the boundary of the second surface to be the same as the subset of vertices on the boundary of the second surface which are also on the boundary of the first surface.

60. The computer program of claim 45 further comprising instructions for causing the computer to:

maintain in a persistent storage a geometrical representation of the first surface.

61. The computer program of claim 46 further comprising instructions for causing the computer to:

store for each node a bounding box on a persistent storage device, and store for each node a list of critical vertices associated with that node on the persistent storage device.

62. The computer program of claim 61 wherein the instructions for causing the computer to store the list of critical vertices comprises instructions for causing the computer to:

store a vertex descriptor for each vertex,

store a parameter value for each vertex, and

store an image value for each vertex.

63. The computer program of claim 62 further comprising instructions for causing the computer to load on demand from persistent storage into memory that portion of the first surface required and remove from memory that portion of the first surface not required.

64. The computer program of claim 63 wherein the computer program causes a tree node to be loaded from persistent storage on demand and to be removed from memory when no longer required.

65. The computer program of claim 64 wherein the computer program causes simplices associated with a tree leaf node to be loaded from persistent storage on demand and removed from memory when no longer required.

66. A computer program, residing on a computer-readable medium, comprising instructions for causing a processor, a data storage system, at least one input device, and at least one output device, to represent a first surface at multiple levels of resolution, the computer program comprising instructions for causing the computer to:

store a grid representation of the first surface, the grid representation being made up of grid cells;

form a mesh representation of a portion of the first surface by triangulating a subset of the grid cells; and

insert the first surface into the model.

67. The method of claim 1 further comprising parameterizing the first surface, wherein the first surface has one or more terminating contours and is tessellated into triangles, wherein parameterizing comprises

if the first surface is not of Euler Characteristic 1:

cutting the first surface into patches, each patch being of Euler Characteristic 1.

68. The method of claim 67 wherein parameterizing comprises

repeating until reaching all terminating contours and all triangles have been processed:

if an annulus of triangles around the boundary of the remaining triangles in a patch cannot be identified because of an obstruction:

refining the obstruction by adding exterior edges until the obstruction is absorbed into the annulus;

constructing a parameterization of the annulus.

69. The method of claim 1, further comprising

filling a hole in the first surface with a tile.

70. The method of claim 69 further comprising parameterizing the first surface, wherein the first surface has one or more terminating contours and is tessellated into triangles, wherein parameterizing comprises

repeating until reaching all terminating contours and all triangles have been processed:

if an annulus of triangles around the boundary of the remaining triangles of the first surface with the hole filled cannot be identified because of an obstruction:

refining the obstruction by adding exterior edges until the obstruction is absorbed into the annulus;

constructing a parameterization of the annulus.

71. The computer system of claim 23 further comprising means for parameterizing the first surface, wherein the first surface has one or more terminating contours and is tessellated into triangles, wherein the means for parameterizing comprises

if the first surface is not of Euler Characteristic 1:

means for cutting the first surface into patches, each patch being of Euler Characteristic 1.

72. The computer system of claim 71, wherein the means for parameterizing comprises

means for repeating until reaching all terminating contours and all triangles have been processed:

if an annulus of triangles around the boundary of the remaining triangles in a patch cannot be identified because of an obstruction:

means for refining the obstruction by adding exterior edges until the obstruction is absorbed into the annulus;

means for constructing a parameterization of the annulus.

73. The computer system of claim 23, further comprising

means for filling a hole in the first surface with a tile.

74. The computer system of claim 73, wherein the means for parameterizing comprises

means for repeating until reaching all terminating contours and all triangles have been processed:

if an annulus of triangles around the boundary of the remaining triangles of the first surface with the hole filled cannot be identified because of an obstruction:

means for refining the obstruction by adding exterior edges until the obstruction is absorbed into the annulus;

means for constructing a parameterization of the annulus.

75. The computer program of claim 45 further comprising instructions for causing the computer to parameterize the first surface, wherein the first surface has one or more terminating contours and is tessellated into triangles, wherein the computer program further comprises

if the first surface is not of Euler Characteristic 1, instructions for causing the computer to:

cut the first surface into patches, each patch being of Euler Characteristic 1.

76. The computer program of claim 75 wherein the instructions for causing the computer to parameterize the first surface comprise

instructions for causing the computer to repeat, until reaching all terminating contours and all triangles have been processed:

if an annulus of triangles around the boundary of the remaining triangles in a patch cannot be identified because of an obstruction:

instructions for causing the computer to refine the obstruction by adding exterior edges until the obstruction is absorbed into the annulus;

instructions for causing the computer to construct a parameterization of the annulus.

77. The computer program of claim 45, further comprising instructions for causing the computer to

fill a hole in the first surface with a tile.

78. The computer program of claim 77 wherein the instructions for causing the computer to parameterize the first surface comprise

instructions for causing the computer to repeat, until reaching all terminating contours and all triangles have been processed:

if an annulus of triangles around the boundary of the remaining triangles of the first surface with the hole filled cannot be identified because of an obstruction:

instructions for causing the computer to refine the obstruction by adding exterior edges until the obstruction is absorbed into the annulus;

instructions for causing the computer to construct a parameterization of the annulus.

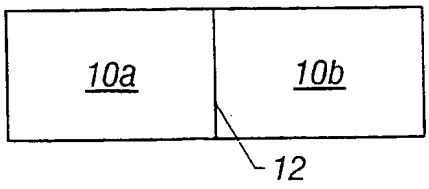


FIG. 1

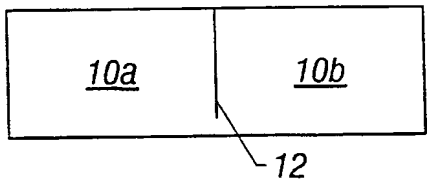


FIG. 2

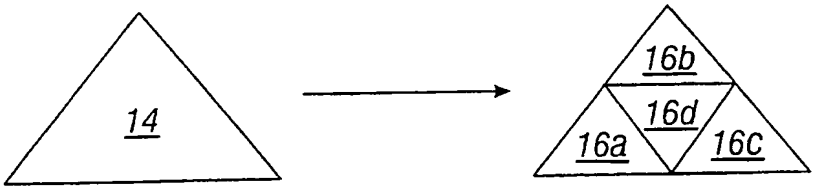


FIG. 3A

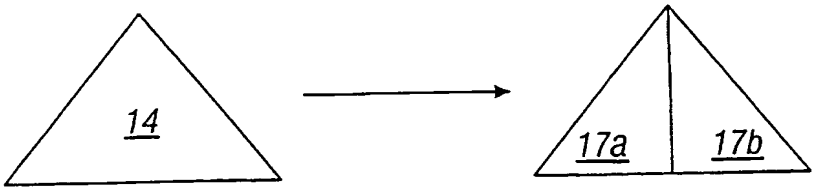


FIG. 3B

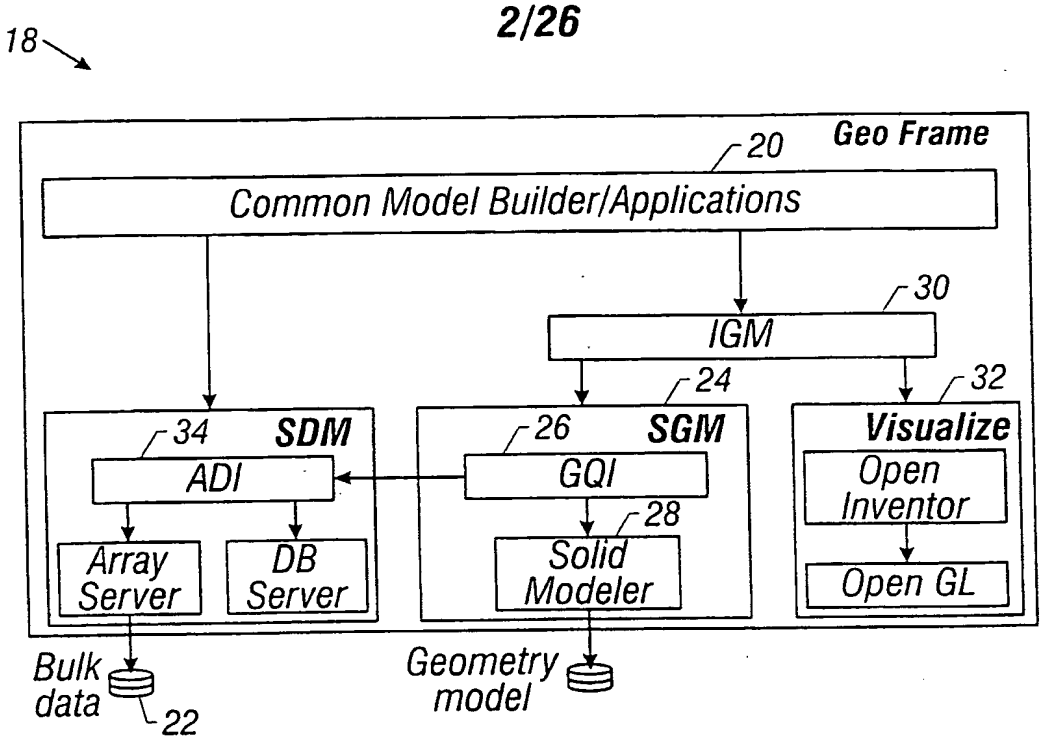


FIG. 4

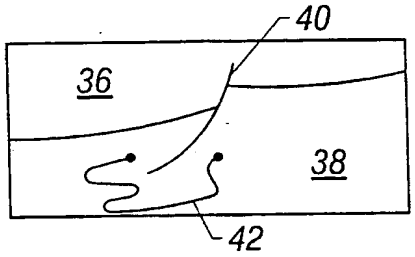


FIG. 5A

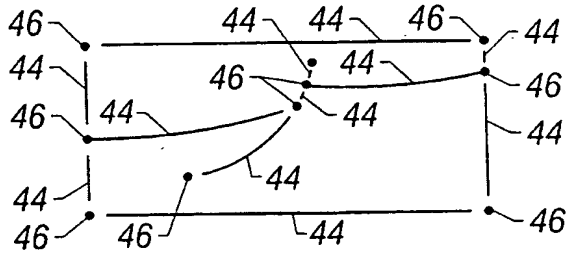


FIG. 5B

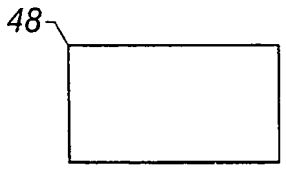


FIG. 6A

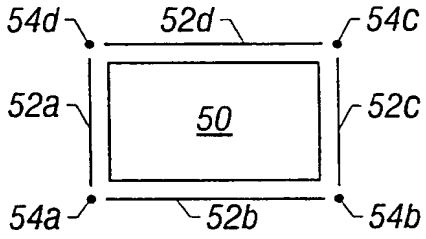


FIG. 6B

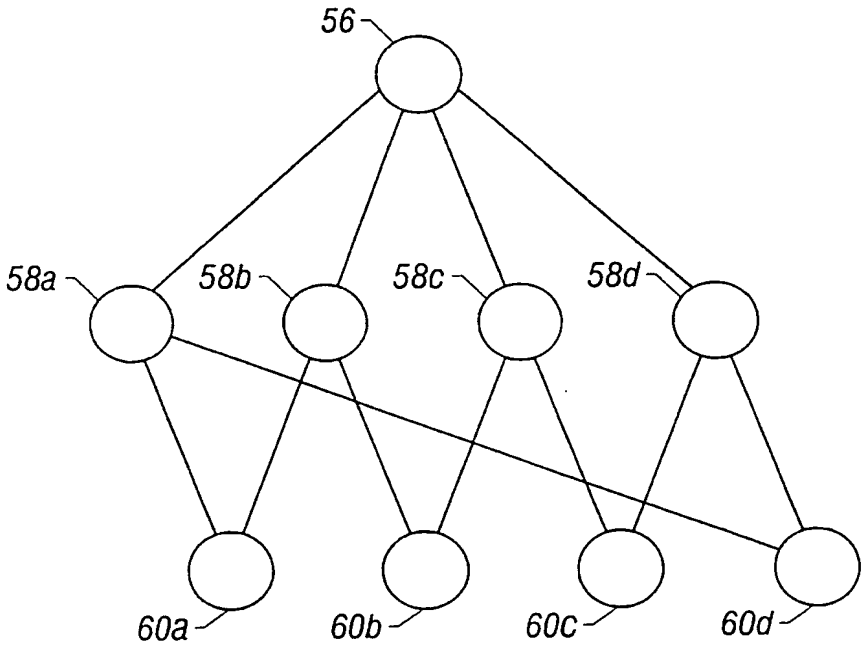


FIG. 6C

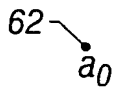


FIG. 7A

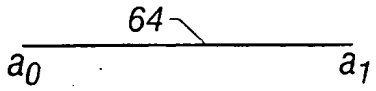


FIG. 7B

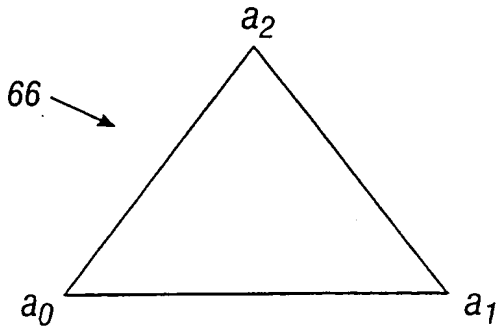


FIG. 7C

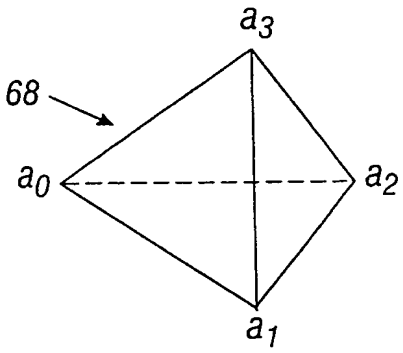


FIG. 7D

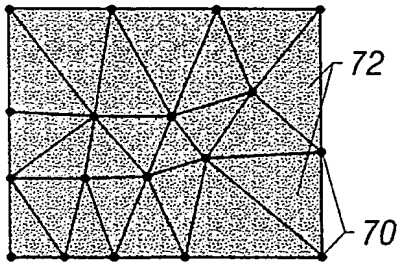


FIG. 8A

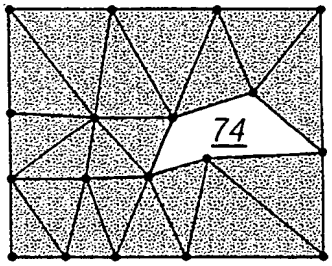


FIG. 8B

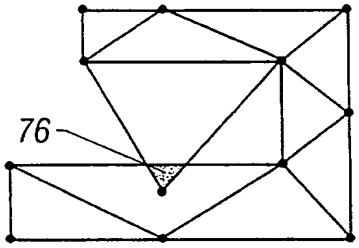


FIG. 8C

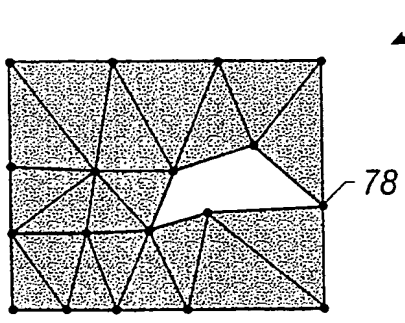


FIG. 9

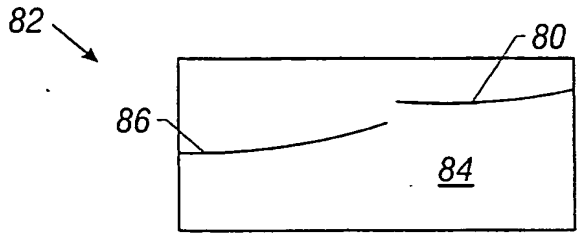


FIG. 10A

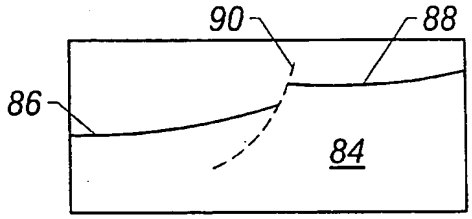


FIG. 10B

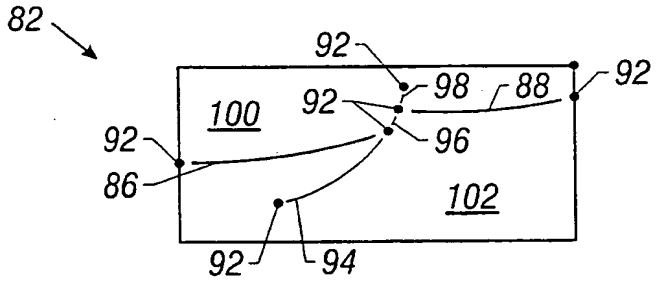


FIG. 10C

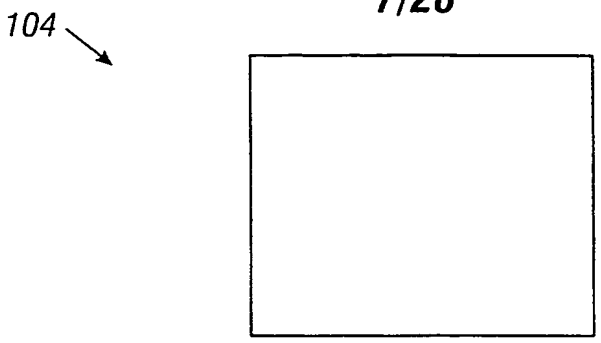


FIG. 11A

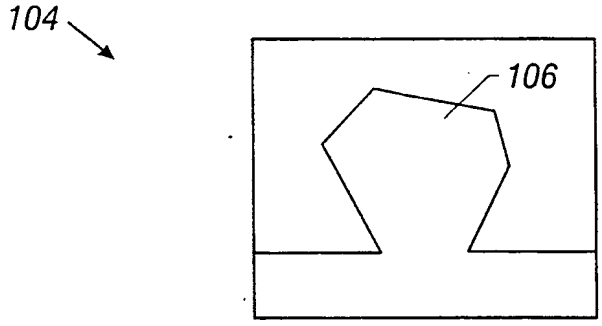


FIG. 11B

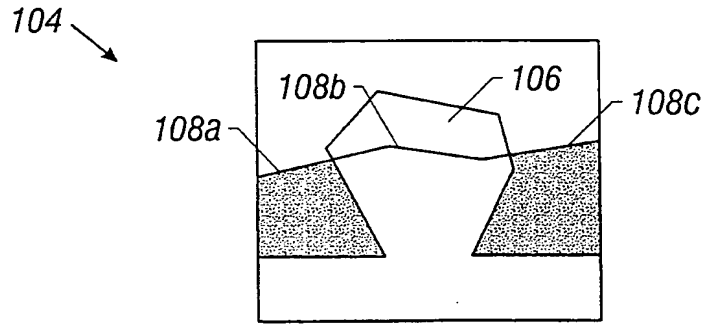


FIG. 11C

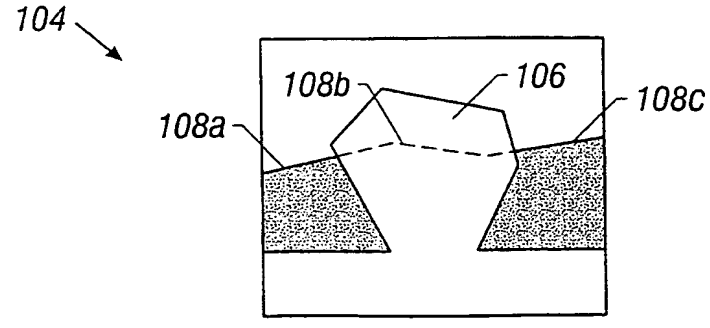


FIG. 11D

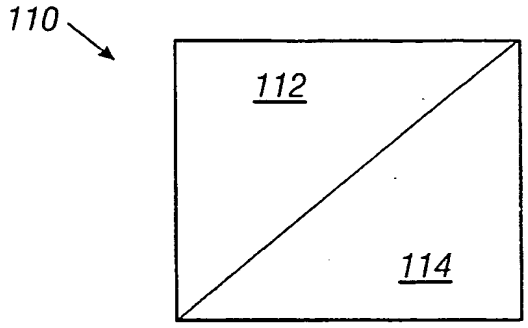


FIG. 12A

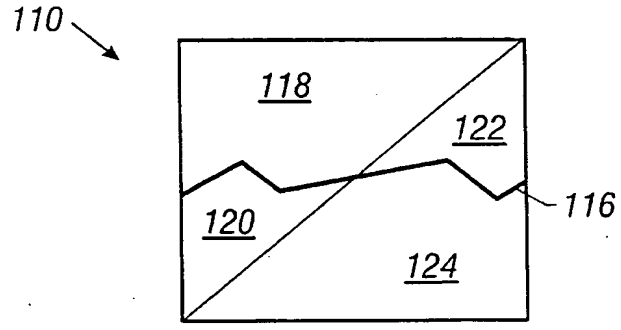


FIG. 12B

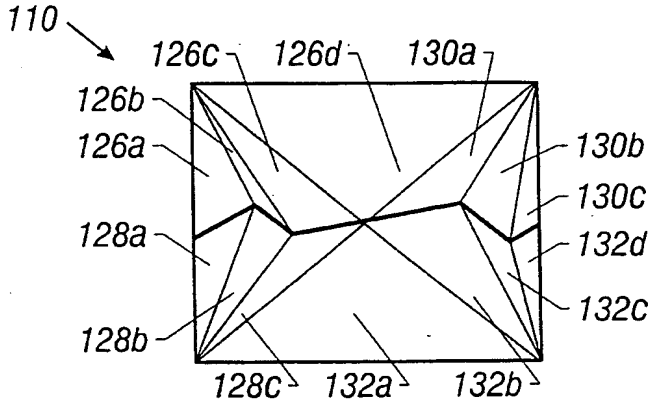


FIG. 12C

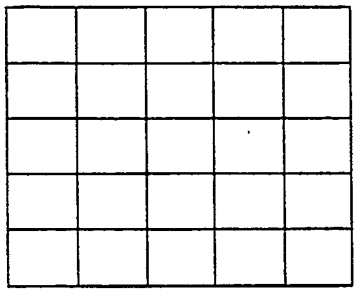


FIG. 13A

FIG. 13B

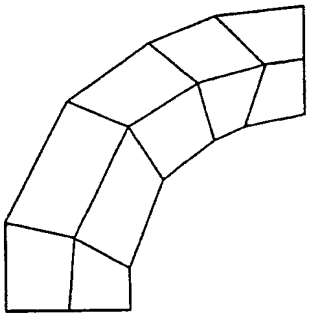


FIG. 13C

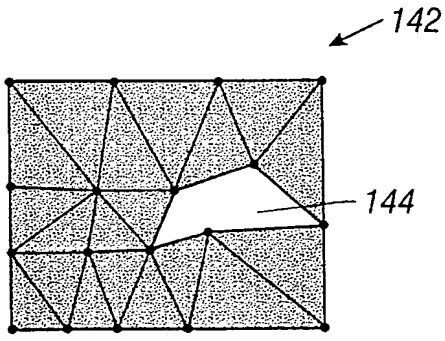


FIG. 14

N x N Grid	Simplices to be rendered	Rendering Time (sec)	Rendering Rate (tris per sec)
100 x 100	19602	0.053	190226
160 x 160	50562	0.099	198000
225 x 225	100352	0.247	204704
318 x 318	200978	0.941	213579
355 x 355	250632	1.152	217562

FIG. 15

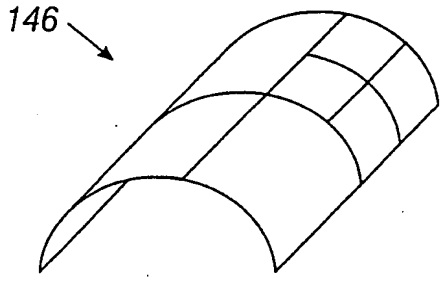


FIG. 16A

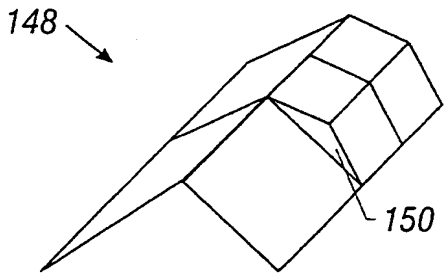


FIG. 16B

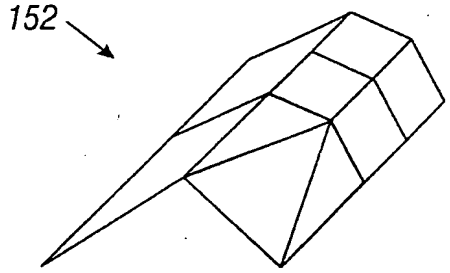


FIG. 16C

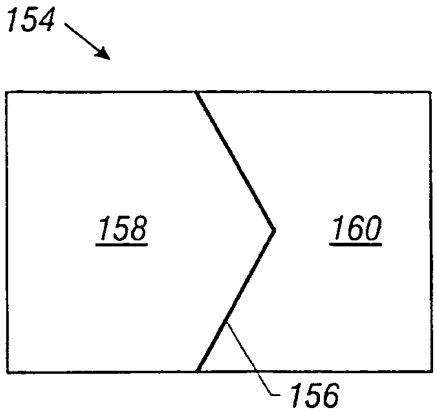


FIG. 17A

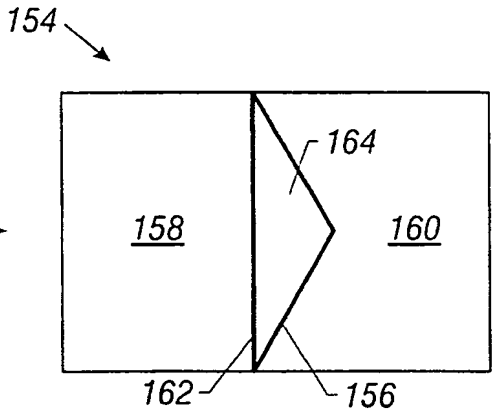


FIG. 17B

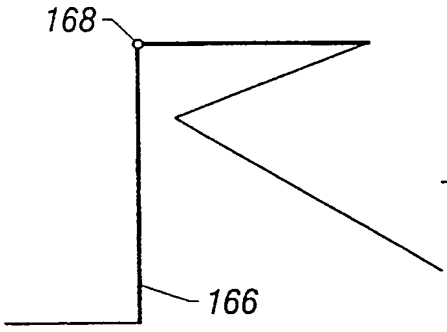


FIG. 18A

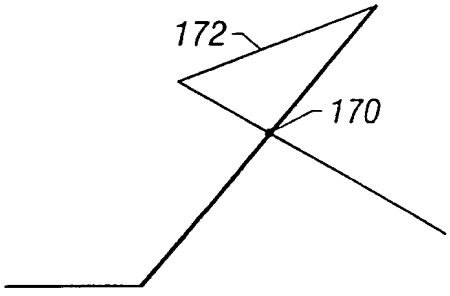


FIG. 18B

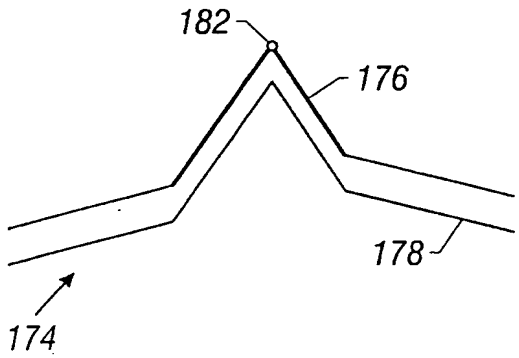


FIG. 19A

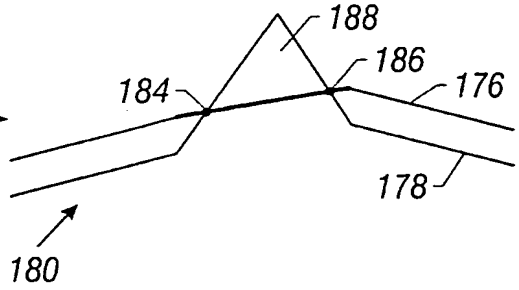


FIG. 19B

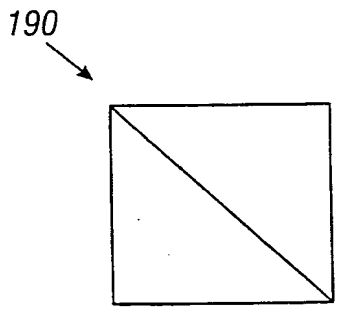


FIG. 20A

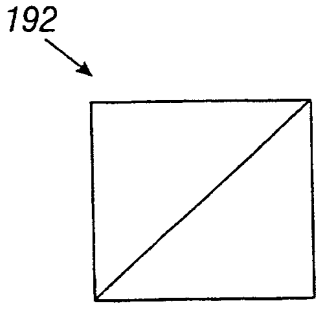


FIG. 20B

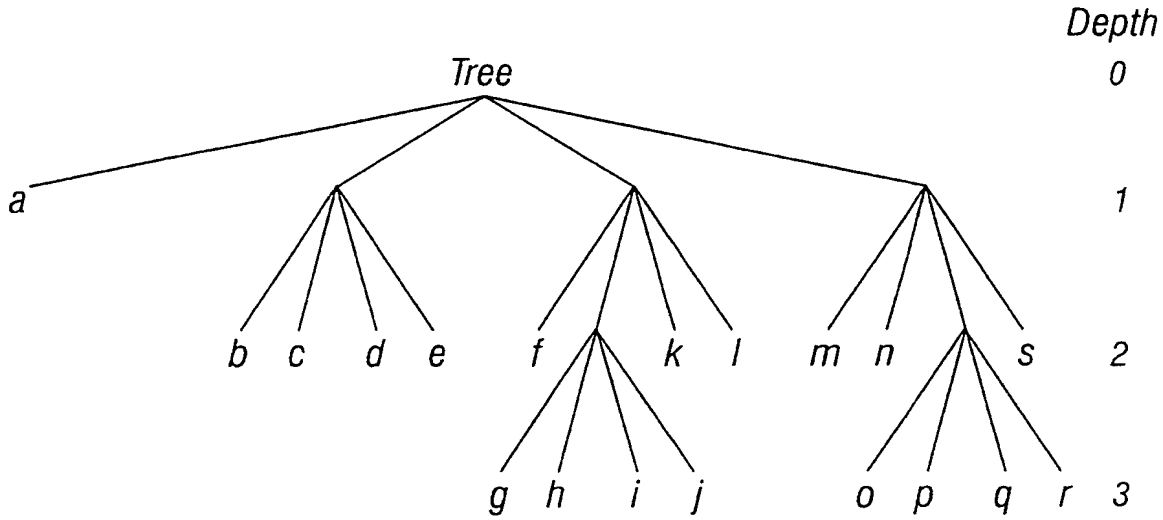

Depth
Tree
0
a
1
b c d e f k l m n s
2
g h i j o p q r
3

FIG. 21

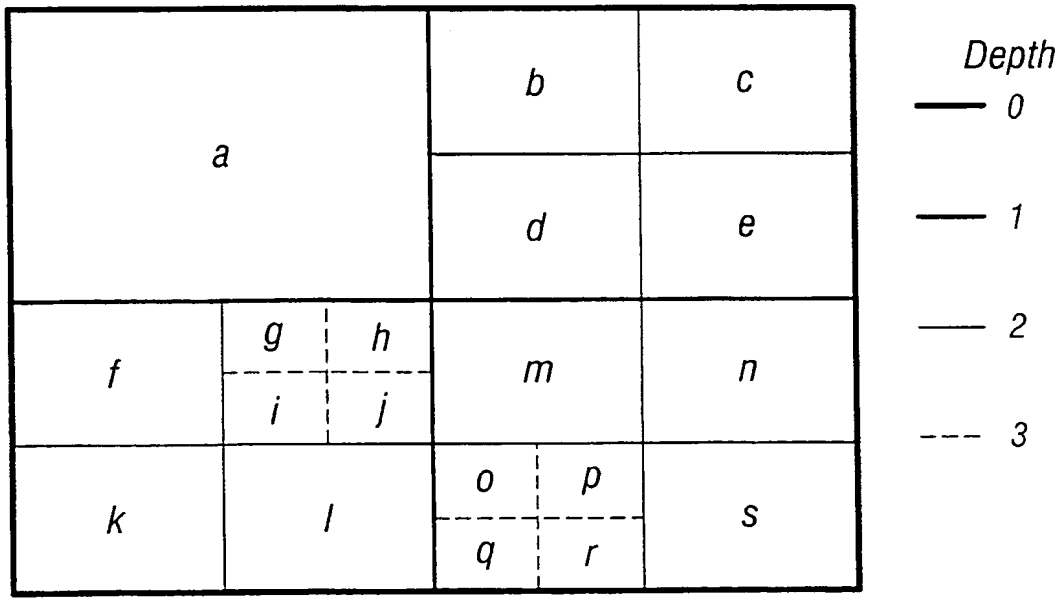

a
b
c
d
e
f
g
h
i
j
m
n
k
l
o
p
q
r
s
Depth
0
1
2
3

FIG. 22

Depth d	Size of grid nxn, $n=2^d$	#bits for key $2d+ceil(\log_2(d+1))$	#leaf nodes 4^d	#nodes $\Sigma_{i=0}^{d} 4^i$
0	1	0	1	1
1	2	3	4	5
2	4	6	16	21
3	8	8	64	85
4	16	11	256	341
5	32	13	1024	1365
6	64	15	4096	5461
7	128	17	16384	21845
8	256	20	65536	87381
9	512	22	262144	349525
10	1024	24	1048576	1398101
11	2048	26	4194304	5592405
12	4096	28	16777216	22369621

FIG. 23

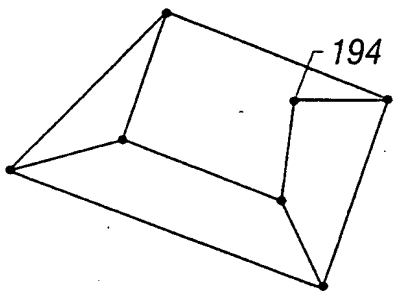


FIG. 24A

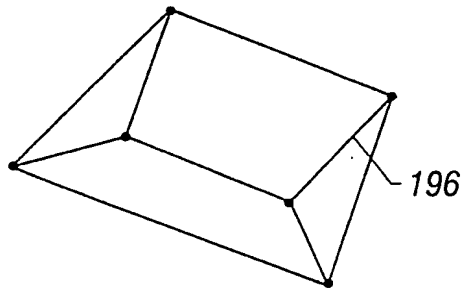


FIG. 24B

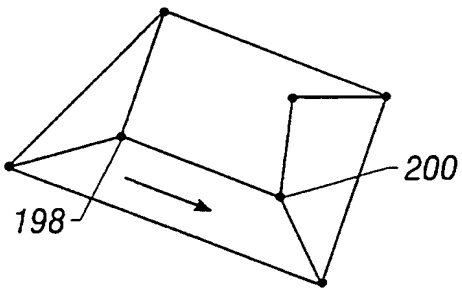


FIG. 25A

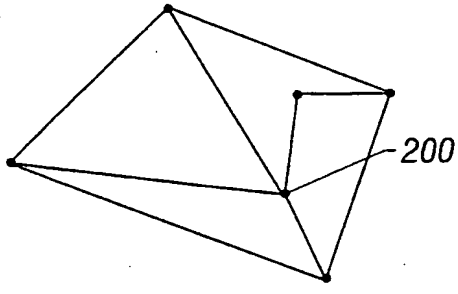


FIG. 25B

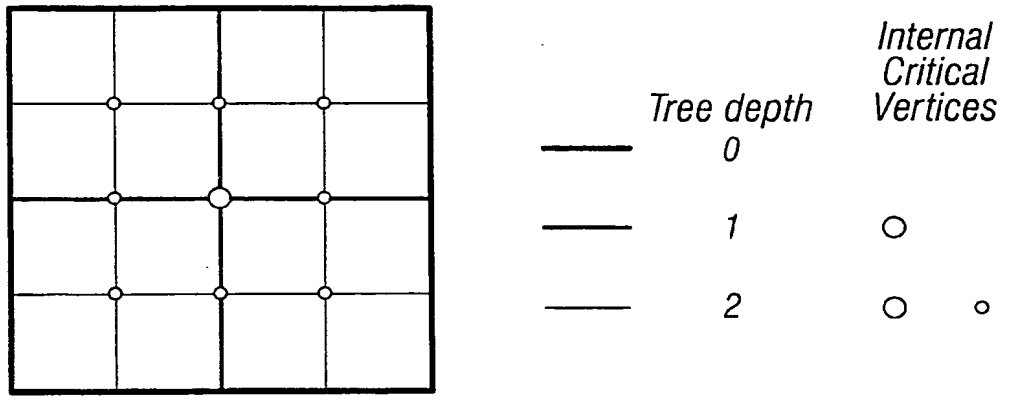

Tree depth
Internal Critical Vertices
0
1
2

FIG. 26

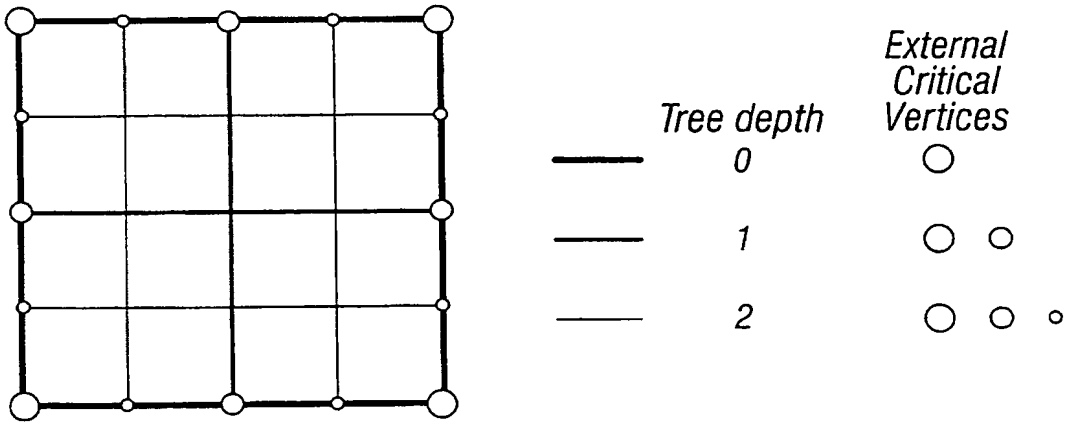

Tree depth
External Critical Vertices
0
1
2

FIG. 27

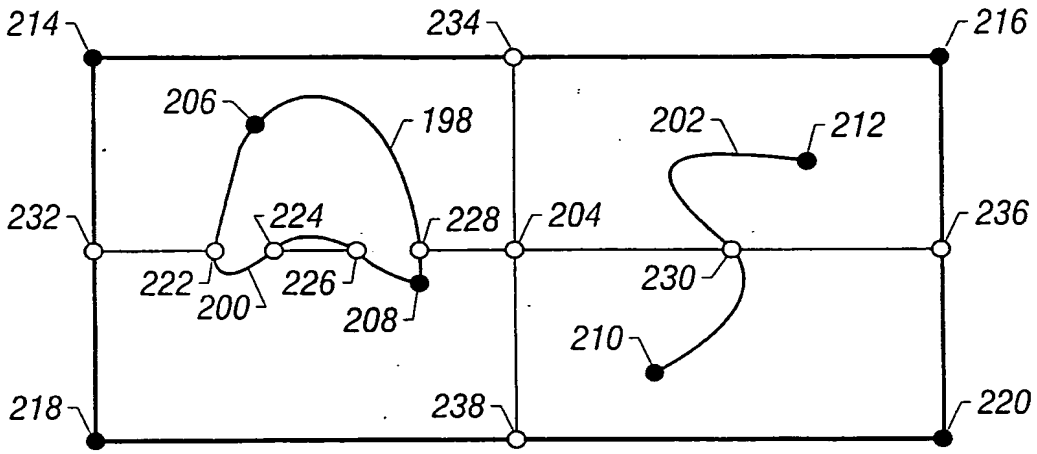


FIG. 28A

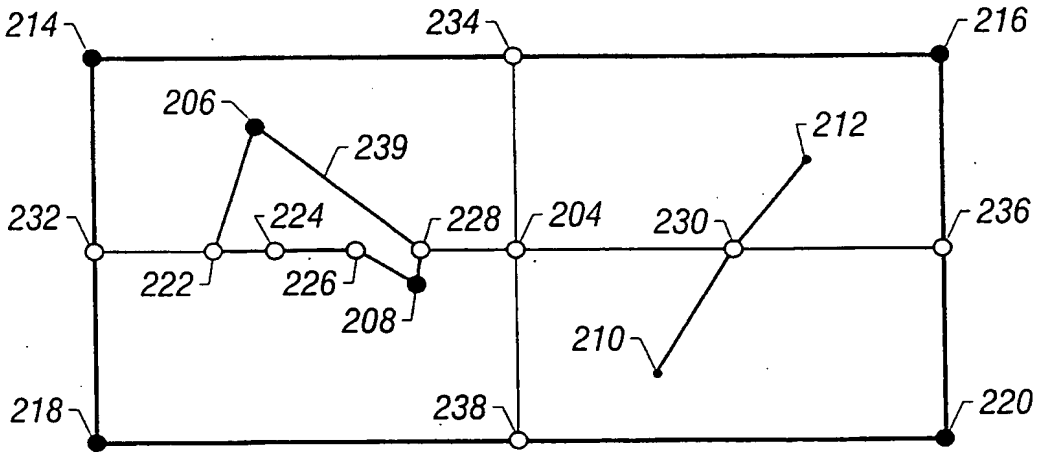


FIG. 28B

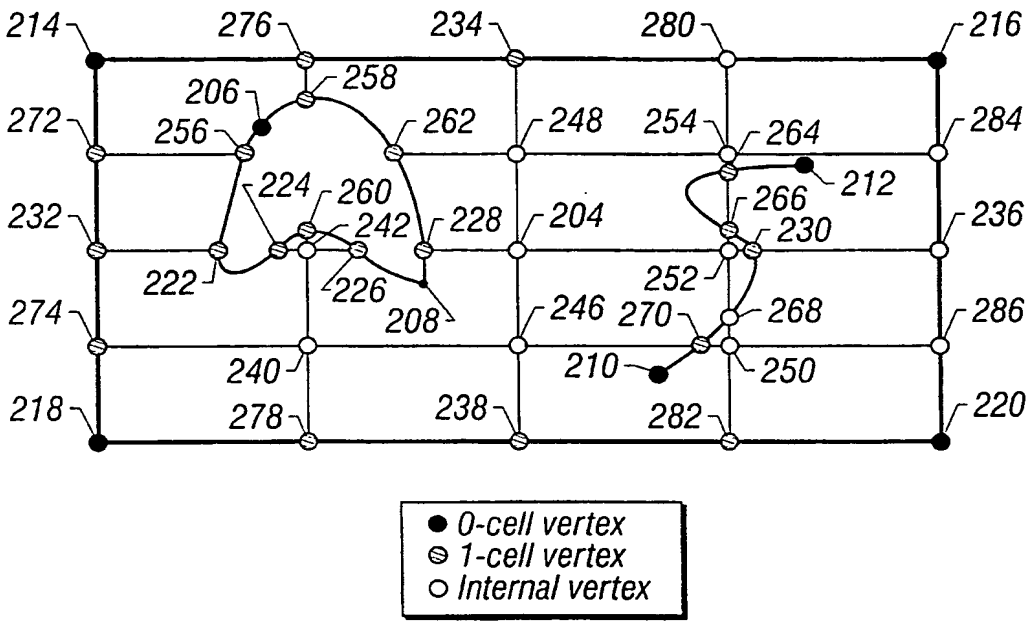


FIG. 29A

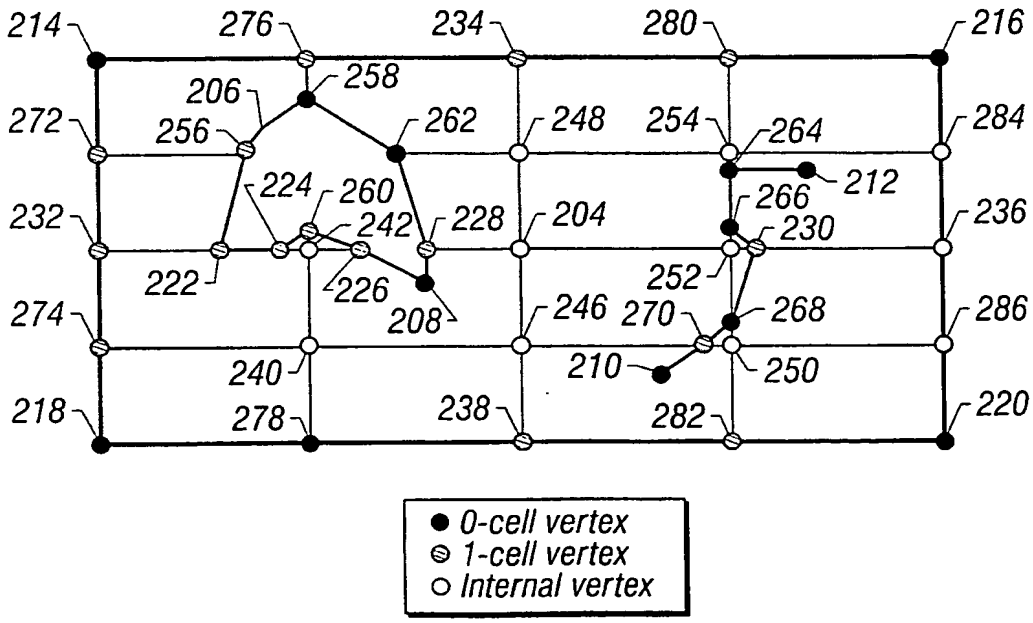


FIG. 29B

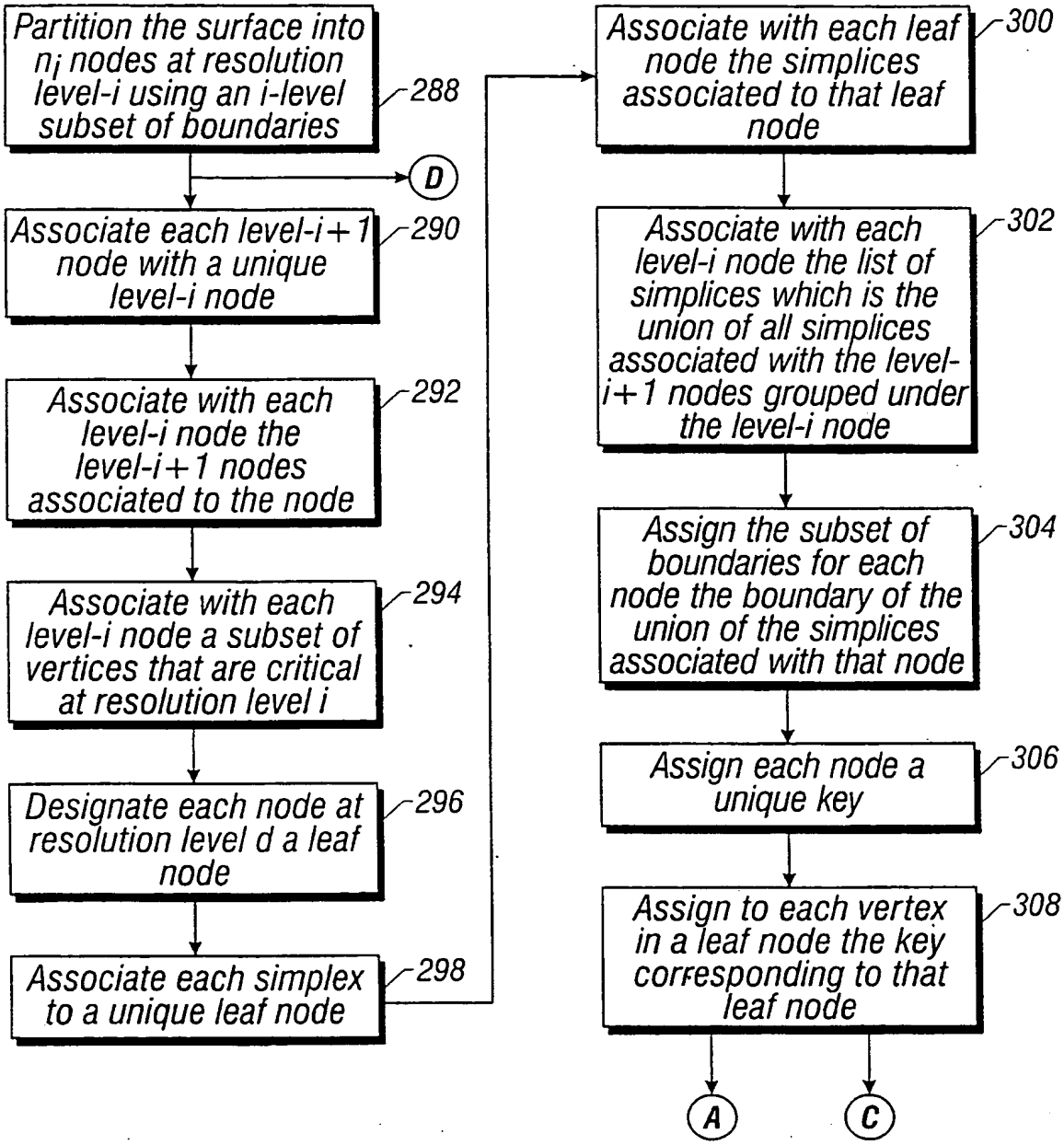

Partition the surface into n_i nodes at resolution level-i using an i-level subset of boundaries
288
D
Associate each level-i+1 node with a unique level-i node
290
Associate with each level-i node the level-i+1 nodes associated to the node
292
Associate with each level-i node a subset of vertices that are critical at resolution level i
294
Designate each node at resolution level d a leaf node
296
Associate each simplex to a unique leaf node
298
Associate with each leaf node the simplices associated to that leaf node
300
Associate with each level-i node the list of simplices which is the union of all simplices associated with the level-i+1 nodes grouped under the level-i node
302
Assign the subset of boundaries for each node the boundary of the union of the simplices associated with that node
304
Assign each node a unique key
306
Assign to each vertex in a leaf node the key corresponding to that leaf node
308
A
C

FIG. 30A

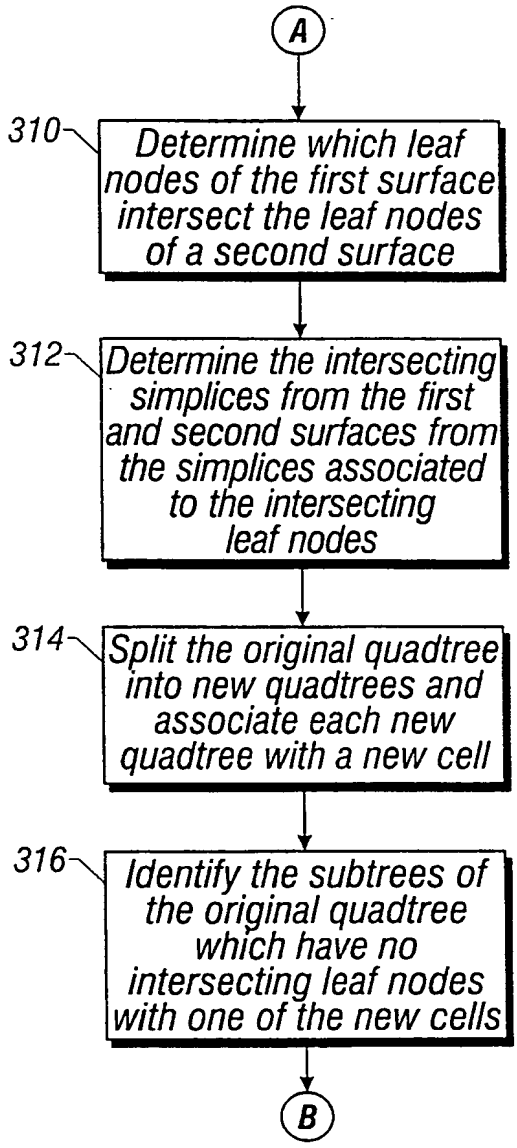

A
310
Determine which leaf nodes of the first surface intersect the leaf nodes of a second surface
312
Determine the intersecting simplices from the first and second surfaces from the simplices associated to the intersecting leaf nodes
314
Split the original quadtree into new quadtrees and associate each new quadtree with a new cell
316
Identify the subtrees of the original quadtree which have no intersecting leaf nodes with one of the new cells
B

FIG. 30B

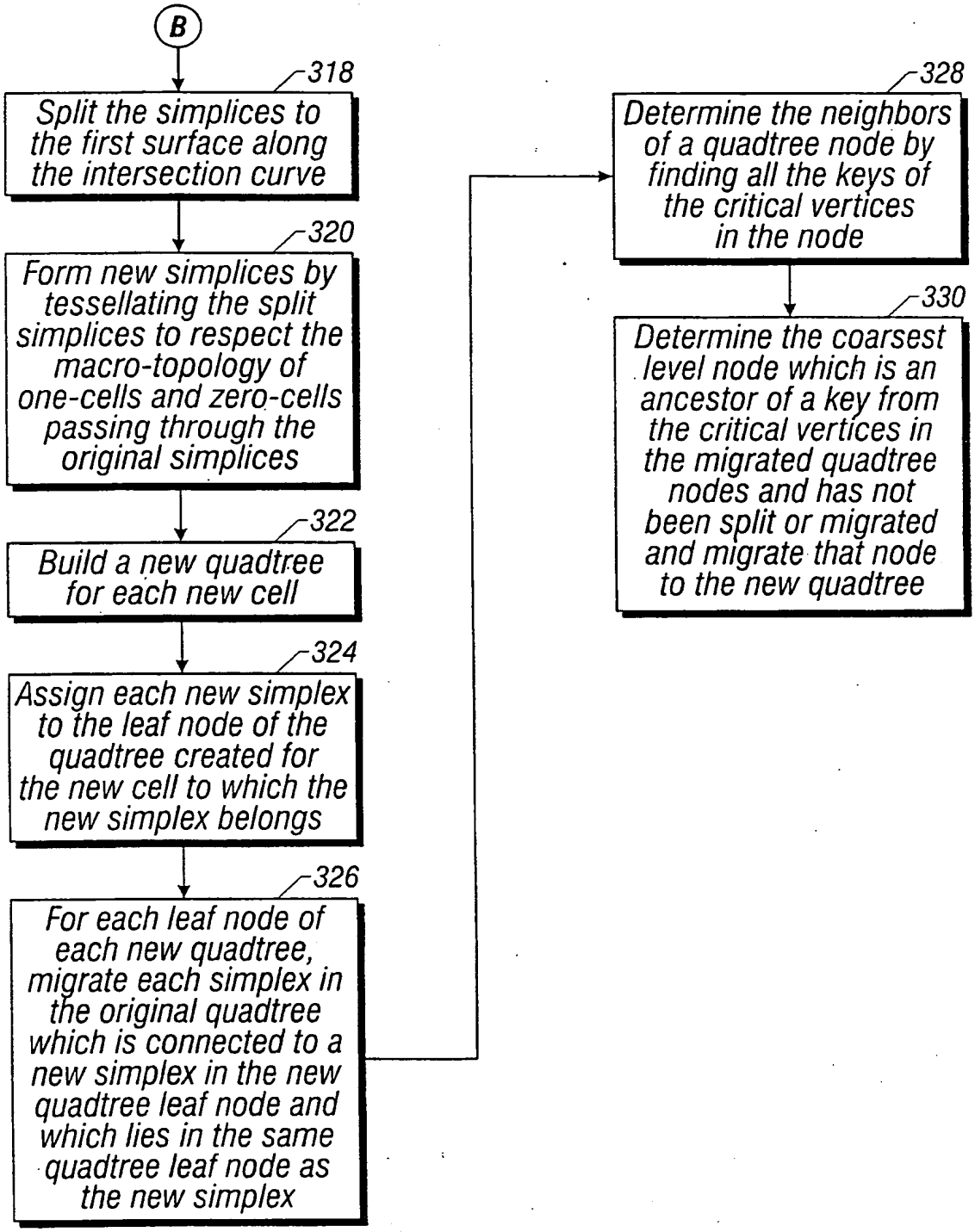

B
318
Split the simplices to the first surface along the intersection curve
320
Form new simplices by tessellating the split simplices to respect the macro-topology of one-cells and zero-cells passing through the original simplices
322
Build a new quadtree for each new cell
324
Assign each new simplex to the leaf node of the quadtree created for the new cell to which the new simplex belongs
326
For each leaf node of each new quadtree, migrate each simplex in the original quadtree which is connected to a new simplex in the new quadtree leaf node and which lies in the same quadtree leaf node as the new simplex
328
Determine the neighbors of a quadtree node by finding all the keys of the critical vertices in the node
330
Determine the coarsest level node which is an ancestor of a key from the critical vertices in the migrated quadtree nodes and has not been split or migrated and migrate that node to the new quadtree

FIG. 30C

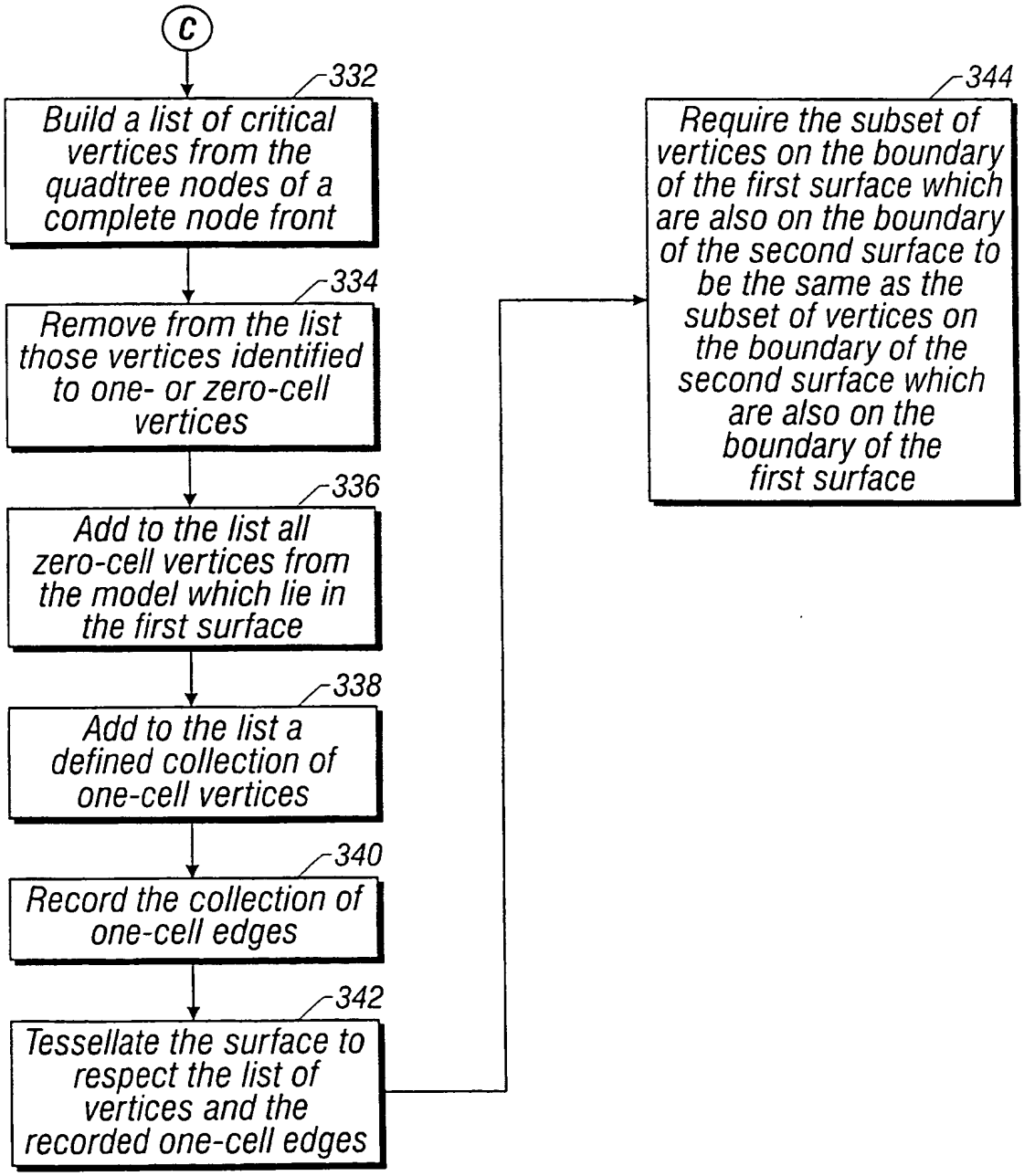

C
332
Build a list of critical vertices from the quadtree nodes of a complete node front
334
Remove from the list those vertices identified to one- or zero-cell vertices
336
Add to the list all zero-cell vertices from the model which lie in the first surface
338
Add to the list a defined collection of one-cell vertices
340
Record the collection of one-cell edges
342
Tessellate the surface to respect the list of vertices and the recorded one-cell edges
344
Require the subset of vertices on the boundary of the first surface which are also on the boundary of the second surface to be the same as the subset of vertices on the boundary of the second surface which are also on the boundary of the first surface

FIG. 30D

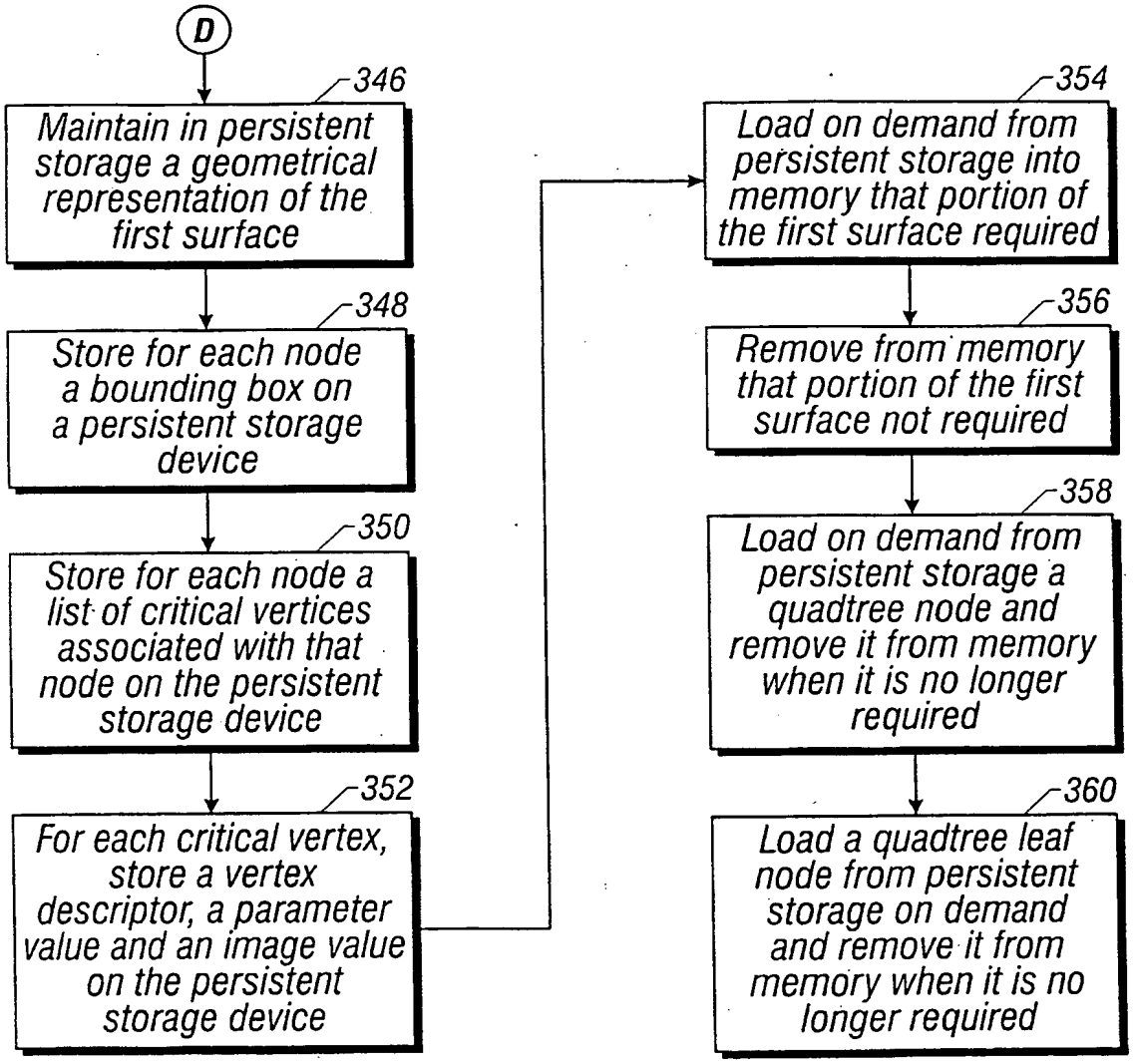


FIG. 30E

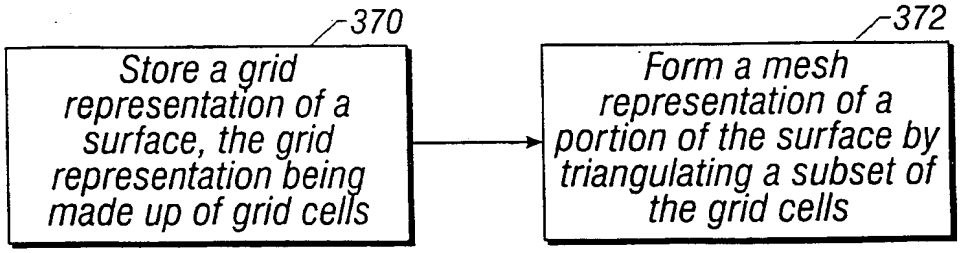


FIG. 33

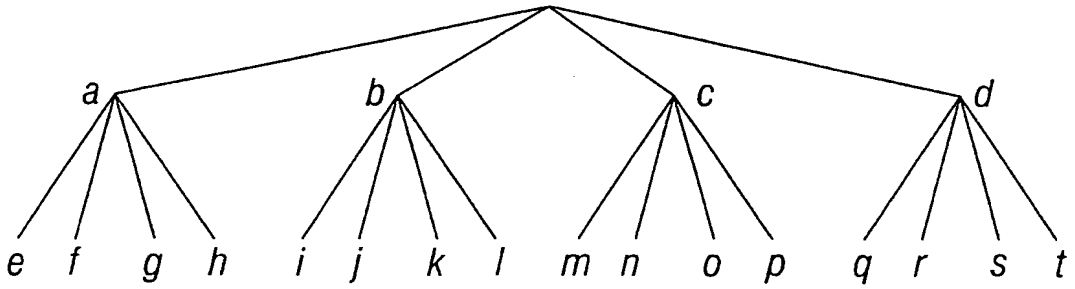

a
b
c
d
e f g h i j k l m n o p q r s t

FIG. 31

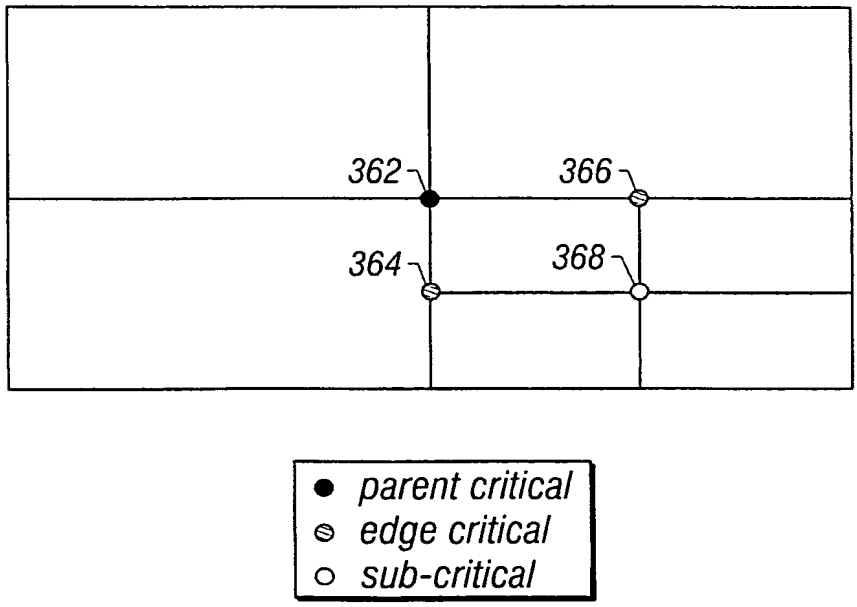

362
366
364
368
parent critical
edge critical
sub-critical

FIG. 32

Surface	Eular Characteristic	g	r
torus	0	1	0
cylinder with zero caps	0	0	2
frustrum	0	0	2
truncated square based pyramid	0	0	2
annulus	0	0	2
disk, square or triangle	1	0	1
sheet with N holes	1-N	0	1+N
cone	1	0	1
cylinder with one cap	1	0	1
punctured sphere	1	0	1
sphere	2	0	0
cylinder with two caps	2	0	0

FIG. 34

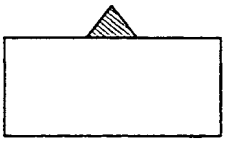


FIG. 35A

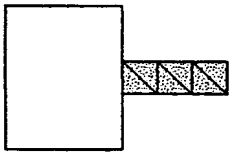


FIG. 35B

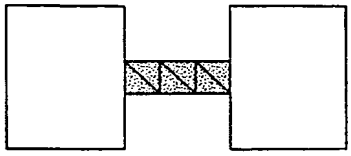


FIG. 35C

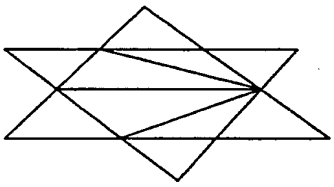


FIG. 36A

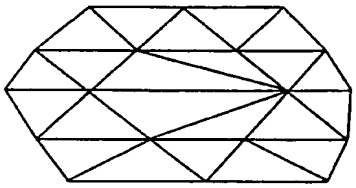


FIG. 36B

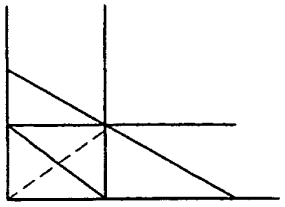


FIG. 37A

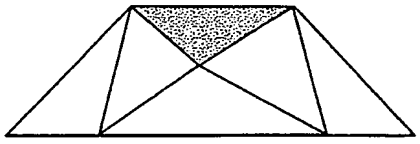


FIG. 37B

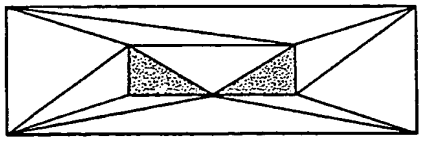


FIG. 37C

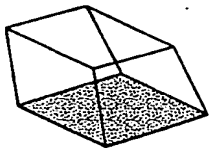


FIG. 38A

FIG. 38B

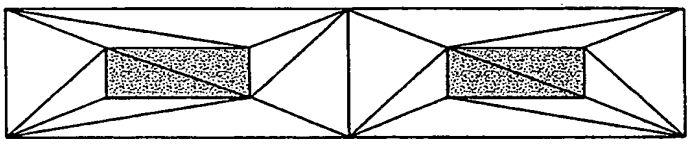


FIG. 39A

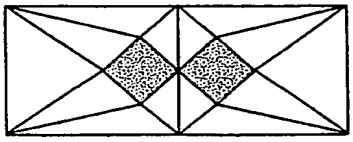


FIG. 39B

FIG. 40A

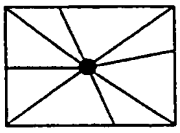


FIG. 40B

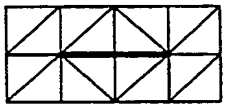


FIG. 40C

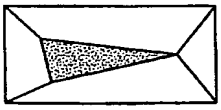


FIG. 41A

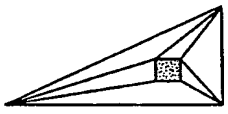


FIG. 41B

INTERNATIONAL SEARCH REPORT

Inte ɔnal Application No
PCT/US 99/22194

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06T17/20

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06T

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	TAYLOR D C ET AL: "An algorithm for continuous resolution polygonalizations of a discrete surface" PROCEEDINGS OF GRAPHICS INTERFACE,CA,TORONTO, ON, 18 May 1994 (1994-05-18), pages 33-42-42, XP002113898	1,22,23, 44,45,66
A	abstract page 34, paragraph 2.1 page 38, paragraph 2.4 page 39, paragraph 2.5 --- -/--	2-21, 24-43, 46-65, 67-78

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication data of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
31 January 2000	08/02/2000

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL – 2280 HV Rijswijk Tel. (+31–70) 340–2040, Tx. 31 651 epo nl, Fax: (+31–70) 340–3016	Gonzalez Ordonez, O

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Intel nal Application No
PCT/US 99/22194

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
A	EP 0 784 295 A (MICROSOFT CORP) 16 July 1997 (1997-07-16) abstract page 2, line 18 - line 27 page 3, line 5 - line 13 page 4, line 45 - line 59 page 7, line 26 - line 54	1-78
A	ECK M ET AL: "MULTIRESOLUTION ANALYSIS OF ARBITRARY MESHES" COMPUTER GRAPHICS PROCEEDINGS (SIGGRAPH),US,NEW YORK, IEEE,1995, pages 173-182, XP000546226 ISBN: 0-89791-701-4 abstract page 176, paragraph 5 page 177, paragraph 7	1-78
A	POPOVIC J ET AL: "PROGRESSIVE SIMPLICIAL COMPLEXES" COMPUTER GRAPHICS PROCEEDINGS. SIGGRAPH,US,READING, ADDISON WESLEY,1997, pages 217-224, XP000765819 ISBN: 0-201-32220-7 abstract page 219, paragraph 3.1 -page 229, paragraph 3.4	1-78

INTERNATIONAL SEARCH REPORT

Information on patent family members

Inte ɔnal Application No
PCT/US 99/22194

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP 0784295 A	16-07-1997	US 5963209 A	05-10-1999
		CA 2194833 A	11-07-1997
		CA 2194834 A	12-07-1997
		CA 2194835 A	11-07-1997
		CA 2194836 A	11-07-1997
		EP 0789329 A	13-08-1997
		EP 0788072 A	06-08-1997
		EP 0789330 A	13-08-1997
		JP 9198524 A	31-07-1997
		US 5966133 A	12-10-1999
		US 5929860 A	27-07-1999